目 录

第二部分 数值技术

第1章 绪　论

使用计算机进行自动计算已经成为工程师、数学家和科学家要掌握的必备技能，虽然计算机可以用来解决很多不同的实际问题，但是在计算机计算和专业学科中存在一个很普遍的核心思想，这个核心就是数值计算。本书是讲述运用数值计算的核心思想及其在MATLAB中的实现。

1.1 专有名词

定义专有名词不仅能使语义纯正，而且可以为读者提供指导。下面部分给出了一些术语的定义，为本书后面的介绍提供语境。

1.1.1 数值计算和符号计算

数值计算是直接对数字进行计算，而不是对方程中的符号进行计算。因此，数值计算产生的结果是数值，而不是数学表达式。当我们求以下式子

1

$$\frac{(17.36)^2-1}{17.36+1}=16.36$$

的值时，就要执行数值计算。相反，符号计算只涉及数学符号的操作，而不管这些数学符号的数值是多少。因此

$$\frac{x^2-1}{x+1}=x-1$$

这个式子是符号计算，当$x\neq-1$时，等式在任何时候都成立。

数值计算的一个重要方面是使用近似值而不是使用精确值。事实上，数值计算现在使用的定义就是对不能用有穷位数表示的量进行估算。考虑分数1/3和常量π:

符号	数值
$\frac{1}{3}$	0.33333...
π	3.14159265358979...

分数1/3是符号公式、表示整数1除以整数3。“...”在除法运算后所得的结果“0.33333...”中表示1/3所得的余数是无穷位的十进制数。当在计算机（或者是手持计算器）上计算“1除以3”时，结果被截成0.3333333、0.3333333333、0.3333333333333333333或其他值，具体结果根据计算设备精确的位数不同而定[⊖]。同样，在计算机程序中使用π时，它也被截成近似值。人为地或者计算机程序中对数值量操作时，比如用0.3333代替1/3、这就叫做数值计算。符号计算包括对x、有理数（如1/3）或超越数（如π）等的无位数限制的操作。

2

例1.1　有理数的数值计算和符号计算

参照以下简单例子:

⊖ 浮点数的二进制表示法在第5章介绍。

$$\frac{1}{2}+\frac{1}{3}+\frac{1}{4}-1=\frac{1}{12} \tag{1-1}$$

符号1/2、1/3、1/4的值分别为0.5、0.33333...和0.25。1/2和1/4都能用有限位数十进制表示，而1/3不能。假设精确到小数点后4位，那么其结果是：

$$0.5000+0.3333+0.2500-1.0000=0.0833 \tag{1-2}$$

实际上，0.0833和1/12 = 0.0833...之间的差别不大，然而在数值上，它们并不相等。数值计算的一个本质特点是符号量在不能整除的情况下用一个近似值来代替，如有理数1/3。

数值计算可以用程序设计语言来运算，如Fortran、Basic、Pascal、C、C++和Java。这些程序设计语言对变量的操作涉及到变量在内存的物理位置。在电路板上，变量被转换成0和1存储在计算机中。二进制（*binary*）（0和1）表示法对应电路元件的“开”和“关”。给一个变量赋值后，计算机将变量值转换成二进制并存储在内存中。并不是所有十进制数都能够精确地表示成二进制数，所以在程序的变量中引入了近似值。变量近似的误差会导致最终结果的误差。一般情况下，近似导致的误差不会很大，但是在某些特殊情况下，很小的数值误差将导致最终结果截然不同，这在第5章将作介绍。

符号计算由计算机程序执行，如Derive®、Macsyma®、Maple®和Mathematica®。MATLAB通过Symbolic Mathematics工具箱支持符号计算，该工具箱使用的是Maple的符号程序。符号计算不必取近似值，而符号计算的速度比数值计算慢。总之，数值计算的速度比符号计算的速度快很多。

符号计算是计算的一个重要方面。然而，本书中介绍符号计算的目的只是为了更好地突出数值计算的优点。

3

1.1.2　数值方法与数值算法

人们通常会认为“数值方法”和“数值算法”是同义词，本文中将通过对概念的介绍来区分这两个定义。数值方法是指对所要执行计算的数学描述；算法是为获得期望结果而进行的一系列的顺序操作。这种定义表明一个好的方法(例如：拥有强大的数学特性)可以用不同的算法实现。而且，根据某些尺度，一种算法可能比其他算法能更好地表达这一方法。

算法与我们的工作生活密切相关。不同的方法可以达到同一目的，不同的算法也可以表达同一方法。例如，为了得到一杯咖啡，可以有好几种实用的方法：用过滤咖啡壶（粗研磨）、用滴漏咖啡壶（中研磨）、浓缩咖啡壶（精研磨）或其他设备等等。如果用滴漏咖啡壶，可以采用两种算法：

算法1

1. 磨碎新鲜的咖啡豆
2. 将过滤器放入支架上并装入磨碎的咖啡进行过滤
3. 冲洗咖啡壶
4. 给咖啡壶中倒入瓶装矿泉水，打开电源煮咖啡
5. 煮咖啡的同时，冲洗咖啡杯
6. 往咖啡杯中倒入煮好的咖啡并加入一些混合啤酒

算法2

1. 取出听装咖啡
2. 将过滤器放入支架，往过滤器中倒入一些咖啡进行过滤
3. 清除咖啡壶里的剩余咖啡
4. 给咖啡壶中倒入自来水，打开电源煮咖啡
5. 煮咖啡的同时，从厨房碗柜中取出咖啡杯进行冲洗
6. 往咖啡杯中倒入煮好的咖啡并加入无脂奶油

[4]

以上两种算法煮出来的咖啡是否有很大的不同呢？这要根据个人口味而定。咖啡的使用目的不同也会有不同的选择：采用算法1煮的咖啡适合与甜点一起饮用或者当作早餐饮用，而用算法2煮的咖啡适合挑灯夜读后休息的时候饮用。

本书第二部分各章中提出了许多数值方法。为了不使读者感到厌烦，每种方法只列出了一种算法，它的选取有三个特点：1.易懂性；2.可靠性；3.高效性。

当确定了一个算法之后，就要使用计算机代码来实现。代码是将算法用某一种计算机语言顺序描述出来。就像不同的算法可以实现同一方法一样，不同的代码可以实现同一算法。第4章重点介绍如何用MATLAB代码有效地实现数值算法。

修改源代码是相当重要的，它是本书每章课后练习的一部分，也是学好数值方法的重要环节。

1.1.3 数值方法与数值分析

数值分析是对数值方法的数学研究。数值分析可以完全脱离计算机而执行，事实上，本书提出的许多技术都是在当前的数字时代之前就已出现的。由于数学家、工程师和科学家对现在某些紧迫的实际问题无法找到合适的解析解，所以他们开发顺序过程来对解析解进行近似计算。手工计算的时代，低能高效的性能要求导致数值分析作为数学的一个分支逐步发展起来。数值分析的重点在于理解数值方法的一般特性，至于如何把这些方法应用到特定的问题中去，它只是数值分析中一个方面。数字计算机的发明及其在科学和工业上的广泛应用更显现出数值分析的重要性。

数值分析得出的结论有助于工作者选择合适的方法和算法，也提供了方法存在的局限性的先验信息，有助于估算计算结果的精确性。根据数值分析得出的结论，程序员可以提高程序的准确性。如果没有数值分析提供的结论，程序员必须通过实验测试各种方法的性能，这不仅浪费时间，也很难总结出编程过程中所获得的有用信息。

[5]

本书的重点是数值技术的实现，而不是分析。分析的结论用于预测数值方法的性能，解释相关方法的不同之处。数值实验用于验证数值方法用在具体的项目上的性能。尽管实验有助于增进对方法的理解，但是实验不能完全取代分析。我们希望通过动手实验，能够增进读者的理论理解能力。有关数值分析的好书，如参考文献[9、10、11、64]，可以作为本书的补充教材。这些书在讨论每种方法时，特别提到了更多的参考书目。这里并不是说数值分析不重要，作者的目的是使用直接的数值试验作为教学手段。

1.2 MATLAB概述

"MATLAB"是"Matrix Laboratory"的缩写。MATLAB的第一个版本是LINPACK和EISPACK库的程序的一个接口，用来分析线性方程组[⊖]。随着MATLAB的演化，除了线性代数外，它还支持许多其他的程序。MATLAB的核心仍然是基于命令行的交互式分析工具。用户可以用类Fortran语言扩展交互环境。交互环境中的程序以命令行的形式执行。

MATLAB用户接口包括下拉菜单和对话框，任何个人电脑使用者对这一接口都很熟悉。菜单命令支持文件操作、打印、程序编辑和用户接口定制。MATLAB的数值计算是通过在命令窗口中输入命令，并不是通过菜单操作进行的。

⊖ MATLAB的第6版用LAPACK替代了LINPACK和EISPACK，其代码可在www.netlib.org上找到。

MATLAB是一个基本的应用程序，它有一个称为标准工具箱的巨大程序模块库。本书介绍的数值方法是由某种工具箱实现的。MATLAB工具箱包括解决实际问题的扩展库，如：求根、插值、数值积分、线性和非线性方程组求解以及常微分方程求解。由于继承了LINPACK、EISPACK和LAPACK的特性，MATLAB对数值线性代数来说是一个高可靠的优化系统。许多[6]数值作业能够用线性代数语言精确地表示。MATLAB和线性代数的密切联系使程序员能够用很短的MATLAB语句来解决复杂的数值作业。

标准工具箱还包括数据可视化的扩展图形库，有简单的点、线和复杂的三维图形和动画。所有的MATLAB程序都可以使用这些函数，这样就可以在所有程序和程序集中分析并生成达到出版质量的图示。对图形的快速访问能有效地提高用户的效率。诊断点有助于调试程序和检验算法是否正确执行。低级的图形函数为自定义图形用户接口的分析代码提供了扩展空间。

除了标准工具箱，可以使用其他的工具箱，如：信号处理、图像处理、优化、统计分析、偏微分方程的求解和许多数值计算的应用。

学生版和专业版　MATLAB作为一种商业产品在不同的计算机平台上都能使用。在个人电脑上，不论学生版或专业版都可以使用。1999年中期，MathWorks(MATLAB的创始人)开始发行MATLAB的Windows和Linux版本。同时，Prentice-Hall也出售MATLAB学生版（student version）。这种学生版的MATLAB包括SIMULINK和符号数学工具箱，它是基于MATLAB 5.3版本而开发的，并且没有矩阵大小的限制。学生版升级到专业版有一次折扣的优惠。

学生版的价钱和一本工程类的教材大致相同，专业版比学生版贵很多。专业版的MATLAB只包括标准工具箱，本书的素材都来自其中的函数。其他的工具箱需要另外购买。而且专业版的MATLAB中矩阵大小仅受限于计算机内存的大小。专业版的MATLAB在新版本发布后可以升级。

如果读者想知道产品更多的信息，可以根据本书提供的地址与MathWorks联系。

1.3　本书的组织结构

本书分为两部分。本章接下来是第一部分，即“MATLAB基础”：这部分让读者对[7]MATLAB的使用有个初步的了解，内容比较多而且易懂。在第一部分中，没有特意地介绍数值方法，只是给出了组成有用算法的基本逻辑结构。第一部分的介绍基于读者对编程语言有一定的基本了解上。对MATLAB已经有所了解的读者可以粗略的阅读该部分。没有接触过MATLAB的读者要仔细阅读第2章到第4章的内容，以便为本书后面章节的阅读打好基础。

1.3.1　MATLAB基础

在学习第2章和第3章时，读者要边读边使用MATLAB，多尝试着做几个实验，学会命令的基本语法（第2章），学会怎样把MATLAB组织成用户定义的函数（第3章）。

第4章，“编制和调试MATLAB程序”，提出了编写高效代码的策略。模块程序是重点而且用实例进行了证明，提出了一种编制文档的风格，描述了调试程序和预测程序缺陷的基本技术。

1.3.2　数值技术

第二部分，“数值技术”，用几个章节介绍基本的数值方法。每章的介绍由浅入深，从简单的算法到复杂的算法。所以，从中间开始读不利于理解。

第5章，“计算中的误差”，为后面章节的算法选取作铺垫。描述了浮点运算的基本特性。

讨论了舍入误差和截断误差，提出执行浮点运算的建议。

第6章，“一元方程$f(x)=0$求根”，从本章开始首次讲述处理具体问题的算法。求根是根据$f(x)=0$计算满足要求的x值的过程。任何只含一个变量的标量方程都能转换为$f(x)=0$的形式。从定点迭代和二分法开始，分别介绍牛顿法、割线法和MATLAB工具箱提供的混合方法。在本章结尾单独介绍多项式的求根。

第7章，“线性代数回顾”，为后面章节的矩阵运算建立基础。如果读者对线性代数有一定的了解，但是没有接触过MATLAB，那么您应该仔细地阅读本章的内容。使用MATLAB解决线性代数问题不仅要求懂得线性代数，而且还要求懂得MATLAB语法。 [8]

第8章，“解方程组”，是许多数值方法的核心部分，也就是求解方程$\boldsymbol{Ax}=\boldsymbol{b}$，其中$\boldsymbol{A}$是已知矩阵（通常为一个方阵），$\boldsymbol{x}$为一个未知列向量，$\boldsymbol{b}$为一个已知列向量。本章从求解$\boldsymbol{Ax}=\boldsymbol{b}$所需的数学条件开始介绍，后面介绍的是高斯（Gauss）消去法。条件数的引入是为了限制高斯消去和后面的替换所产生的误差。第8章也介绍了用LU分解和Cholesky分解求解$\boldsymbol{Ax}=\boldsymbol{b}$方程。在MATLAB中，求解线性方程组使用“\”符号，它称为“反斜杠符号”，这种方法是针对矩阵A的不同特性而使用LU分解、Cholesky分解或者QR分解方法。在MATLAB中，反斜杠符号的功能非常强大，第8章的目的在于介绍如何正确使用反斜杠符号。本章末尾将讨论非线性方程系统的求解

第9章，“数据的最小二乘曲线拟合”，介绍了最小二乘法在数据建模中的应用。本章首先介绍（x、y）数据向一条直线拟合的简单算法，然后讲述了更一般的对一个数据集的任意（并非线性）函数组合的拟合问题。本章的一个重点是解释了如何使用MATLAB中内置的反斜杠操作符，由QR分解来求最小二乘解。本章也介绍了多元最小二乘拟合。

第10章，“插值”，介绍了离散数据的插值。这里考虑了两种基本方法：任意次数的多项式插值和分段多项式插值。本章还介绍了一维多项式和样条插值的不同算法，并使用MATLAB内置的`interp`函数来描述其算法。

第11章，“数值积分”，讨论了没有初等函数解析式（closed-form）的定积分数值逼近的不同方法。梯形法则和Simpson法则是作为一般的程序推导和实现的。这里解释了高斯求积法的基本理论，给出了任意次数的复合高斯求积法的程序。本章还讨论了自适应求积法，并示范了`quad`和`quad8`程序。

第12章，“常微分方程的数值积分”，描述了初值问题求解的不同方法。其中，Euler法揭示了基本的概念，由此展开了一系列更精确和有效的方法，其中以自适应步长的Runge-Kutta法为最好。此方法首先推导了一阶常微分方程（ODE）的应用，然后扩展到联立方程组和高阶ODE的应用。 [9]

1.3.3 MATLAB程序的交叉引用

书中用方框标识了所用的MATLAB程序清单以及对程序的描述语句。每章末尾的表格中列出了本章所写的程序，而且在本书的末尾还列出了3个索引。第一个索引列出引用MATLAB内置函数的页码，第二个索引列出程序的页码，第三个索引是本书的交叉引用。

1.4 练习的难易级别

根据读者对本课程掌握深度的不同，课后练习的难易程度也不等。显然，对数值方法越精通，做课后题就越容易。本书作者的初衷是便于布置课后练习和自学者能更好地学习。这

一难易级别的确定模拟了爱斯基摩人与筏夫对渡过不同的河流分类的方法[⊖]。

作业的级别分类如下所示:

1级 对MATLAB概念的直接应用，虽然它的级别最低，但是概念性的知识是不容忽视的。

2级 书中列出方法新的应用、有的要做一定的修改。要融入自己的思想以便和标准方法一起使用。

3级 充分理解书中列出的方法后，要对它们作些必要的修改，由于错误的应用将导致潜在的严重错误、所以最好能够验证其结果的正确性。

4级 要得到解决方法必须对书中提出的技术进行扩充，而且解法有可能一开始就是错的，所以思维灵敏是特别重要的。外部数据的引入，能够开发出更有用的程序。同时，要找到一种验证结果正确性的方法不是一件简单的事情。

[10]

和前面提到的河流分等级一样，练习的分类可以在级别后面增加加号或者减号。如3－表示比3级略容易、4+表示比4级稍微难一点。

对河流分等级熟悉的读者可能已经注意到，书中没有提到等级5和等级6。它们分别对应"只有专家才能进行无错操作"和"专家的极限，任何人都可能产生错误"。在问题求解中、任何级别中的错误、特别是级别5和级别6的错误都是致命的。本书没有必要介绍级别5和级别6。

本书引用河流分等级系统有两个原因。第一，作者喜欢乘船漂流。第二、随着划船这项技术的提高和先进设备的使用，以前的4级现在可能降到4－甚至3+。作者希望随着数值软件、比如MATLAB的不断发展和数值分析的不断进步，现在的难题在不久的将来易于管理或者能够用程序来实现。

盲目的乐观不可取、初学者可能会觉得自己已经什么都能做了，更坏的是由于功能强大的软件、奇妙的图形和高速计算机的使用、初学者会步入误区、他们觉得自己的程序按照正确的步骤执行就表示已经理解了计算过程。作者希望学习者能够把自己的重点放在某个特定的水平上、并在更高水平的应用问题中发现自己的潜能。作为一个数值分析的老师、应该鼓励学生深入学习、在必要的时候给予帮助。

[11 ~ 14]

⊖ 想知道更多关于American Whitewater Affiliation International Scale of River Difficulty的信息，请登录www.awa.org。

第一部分　MATLAB基础

第2章　MATLAB的交互计算

本章将介绍MATLAB的基础知识：命令的输入、变量的定义和使用、二维和三维图形的创建。这些操作要一次键入一条命令，交互地进行。很多复杂的数值作业用几条简明的命令就能够实现，所以MATLAB的交互计算是非常高效的。交互命令都能够在程序中使用。本章是学习MATLAB的基础，图2-1总结了本章的主题。

本章主题

1. **运行MATLAB**
   演示MATLAB作为表达式求值的交互过程，介绍用户自定义变量、内置变量和内置函数，详细介绍了在线帮助系统。
2. **矩阵和向量**
   MATLAB中的所有变量，包括标量和向量都被看成矩阵。本节介绍如何定义和操作矩阵变量，介绍如何使用**冒号运算符**和下标来引用矩阵中的元素，以及删除矩阵中的元素。
   向量化操作是对向量元素或矩阵元素的数学操作，对矩阵元素的操作采用向量声明和数组操作数。
3. **附加类型变量**
   通过实例讲解MATLAB对复数、多项式和字符串的操作。
4. **管理交互环境**
   本节介绍如何管理、存储和在交互环境下重新载入使用的变量，描述了在MATLAB中将数据从磁盘文件读入的过程。
5. **MATLAB中的画图**
   介绍产生线、面和轮廓图形的函数，介绍通过坐标和图例对图形的注释。

图2-1　第2章的主题

2.1　运行MATLAB

不同种类的计算机安装和启动MATLAB的方法各不相同。运行MATLAB后，会出现下拉菜单，以及一个或者多个命令窗口。这个窗口可以用作输入命令并且显示输出结果，下拉菜单用于管理文件、帮助命令窗口编辑和控制显示输出。与其他计算机程序的菜单相比，MATLAB的菜单功能相当少。事实上，在MATLAB中的所有数值计算都是通过命令行执行，而不是通过菜单进行的。 [15]

学习MATLAB的一个有效方法就是边学边用，通过输入本书中的命令来熟悉各种命令。启动MATLAB时，出现命令窗口并显示命令提示符

```
>>
```

[16]

如果使用的是MATLAB的教学版，命令窗口中的命令提示符是

```
EDU>
```

出现命令提示符，表示可以向MATLAB命令窗口输入命令了，紧接命令提示符后输入命

令，用回车键结束输入。对不同的命令，MATLAB会有不同的响应，有时在命令窗口出现文本信息；当需要输出图形时，MATLAB新建一个窗口输出图形。执行完一个命令，会另起一行出现新的命令提示符，这种模式就是MATLAB的交互性（MATLAB对每行命令都会做出响应）。在有些情况下，MATLAB响应的只是在下一行出现的命令提示符，这表示MATLAB等待用户重新输入命令。

排版约定 本书的所有命令都用等宽字体，比如

```
>> help matfun
```

当命令窗口中出现命令提示符请求用户输入命令时，这时作为写输入是有效的，用户可直接将命令输入到命令窗口中。当文字的目的是讨论一个函数的输入输出参数时，不会出现命令提示符，而且参数名字用斜等宽字体显示，如

```
ones(nrows,nccls)
```

这表示ones函数带两个参数：*nrows*和*ncols*。这里没有出现命令提示符，所以是不可写的。ones函数的命令提示符的写法是

```
>> ones(3, 5)
```

有时，命令行后面的注释以"%"开始，它后面的所有字符都是注释的内容，比如

```
>> x = sqrt(-4)   % MATLAB automatically computes with complex numbers
```

在命令提示符后面输入命令时，可以不写注释语句。

变量（在2.1.2节介绍）以等宽字体显示，先定义，后使用：

```
>> nr = 2
>> nc = 2
>> ones(nr,nc)
```

17

交互命令的输出结果同样以等宽字体显示，它紧接在命令行的后面。命令、变量和输出结果都是模拟MATLAB在计算机屏幕上的输出。因此，上面命令的输出将是

```
>> nr = 2
nr =
     2


>> nc = 3
nc =
     3

>> ones(nr,nc)
ans =
     1   1   1
     1   1   1
```

前面为了紧凑，删除了多余的命令显示行。

2.1.1 MATLAB用于表达式计算

MATLAB能用于计算简单的数学表达式。向窗口中输入表达式后按回车键就可以进行计算：

```
>> 2 + 6 - 4
ans =
     4
```

在用户没有把表达式的值赋给用户自定义的变量时，系统将自动把所得的结果赋给系统变量*ans*。为便于以后计算使用，系统保存*ans*的结果：

```
>> ans/2
ans =
    2
```

除了能计算表达式的值，我们还可以把表达式的值赋给某个变量，如：

```
>> a = 5
a =
    5
>> b = 6
b =
    6

>> c = b/a
c =
    1.2000
```

18

可见，c代替ans用于存储表达式的值。

MATLAB有许多内置函数和预定义变量。普通的三角函数也包含在函数集中。函数的参数紧跟在函数名后的括号中，如

```
>> sin(pi/4)
ans =
    0.7071
```

内置变量pi的数学表达式用于计算输入参数的值。所有三角函数的单位都采用弧度制，不采用“度数”制。

不论是内置变量还是用户自定义变量，在命令提示符后面输入变量名，回车后都能看到变量的值：

```
>> pi
ans =
    3.1416
```

我们并不关心计算结果的中间值，只关心最后的返回结果。在命令行的末尾加上分号，就不会显示输出结果：

```
>> x = 5;
>> y = sqrt(59);
>> z = log(y) + x^0.25
z =
    3.5341
```

分号不是必需的，使用分号表示不需要马上输出计算结果。为了节省空间，一行可以输入多个表达式，这就要求用分号或逗号把每个表达式区分开。使用逗号能够在一行输入多个表达式，但是这样做会显示计算过程的中间结果：

19

```
>> a = 5;  b = sin(a),  c = cosh(a)
b =
    -0.9589

c =
    74.2099
```

2.1.2 MATLAB变量

前面已提到过，变量要先定义，然后才能使用。当一个变量未被定义时，在给它赋值时会创建这一变量。而且，变量的值可以通过后面的赋值语句来改变：

```
>> t = 5;
>> t = t + 2
t =
    7
```

等号右边的任何变量必须在前面被定义过才能使用。语句

```
>> x = 2*z
??? Undefined function or variable 'z'.
```

本例中由于变量'z'在前面未被定义，导致计算错误。

在MATLAB 5中，变量名的最大长度为31个字符。变量名区分大小写，x和X被认为是两个不同的变量。变量名必须以a-z或A-Z开头，后面的字符可以是任何字符组合，包括下划线"_"。长变量名有利于在文献编制程序中使用（参见4.1.3节）。下面的语句用来进行常见的几何计算：

```
>> radius = 5.2;
>> area = pi*radius^2;
```

[20] 这里如果使用长变量名会令人很厌烦。

2.1.3 内置变量和函数

MATLAB中使用的内置变量名的长度都很短。如ans变量，一旦某个表达式的值没有赋给一个已定义变量，它将自动赋给ans。表2-1中列出了MATLAB中的内置变量及其含义。内置变量由内置函数使用，所以用户最好不要给内置变量重新赋值。i，j是一个例外，在矩阵下标中常用到i和j（参见2.2.2节）。

表2-1 MATLAB中的内置变量

变量名	含义
ans	当表达式的值未赋给某个变量时，系统自动将它赋给ans
eps	浮点数精度
i,j	单位虚数，i = j = $\sqrt{-1}$
pi	π,3.14159265...
realmax	最大正浮点数
realmin	最小正浮点数
Inf	∞，比最大正浮点数(realmax)大，是1 / 0的值
NaN	不是一个数（例如，0 / 0的值）

内置变量eps称为机器精度，它用于构造公差。本书将在第5章讨论机器精度问题。内置变量realmax、realmin、Inf和NaN用于浮点计算中的异常处理。realmax和realmin是以双精度数存储在计算机中的最大值和最小值[⊖]。变量Inf（"infinity"）和NaN（"not a number"）在出现浮点运算异常时使用。下面的计算使用了变量Inf和NaN：

⊖ 第5章讨论浮点数精度及其与realmax、realmin的关系。

```
>> x = 0;
>> 5/x
Warning: Divide by zero
ans =
     Inf

>> x/x
Warning: Divide by zero
ans =
   NaN
```

21

我们会在第5章深入介绍变量eps、inf、realmin和realmax。

MATLAB包含许多内置函数，以至经常会出现用户使用内置函数作为变量名的情况。例如，用户可能不会知道gamma内置函数用于计算

$$\Gamma(x)=\int_0^\infty t^{x-1}e^{-t}\,dt$$

下面的语句

```
gamma = 1.4
```

就会创建一个叫做gamma的变量，这也是合法的，但这将导致在本次与MATLAB的交互中，gamma函数不可用。

2.1.4 函数和命令

有必要区分函数和命令，它们都在命令窗口中输入。函数有输入变量和输出变量。如：

```
>> y = sin(pi/6)
```

函数sin的输入变量是pi/6，输出变量是y。而命令的使用，如help命令，在命令和变量之间用空格隔开。如：

```
>> help sin
```

总之，命令是对MATLAB本次交互的操作，而函数是对MATLAB中变量的操作。

命令和函数的区别不是绝对的。命令也能够像函数一样使用，只是变量名要以字符串的形式输入，如

```
>> help('sin')
```

但是，函数不能像命令一样使用。如下表达式

```
>> sin pi
ans =
   -0.8900 -0.9705
```

出现了错误的结果。

22

学习MATLAB主要是学习函数的使用。由于只有少量的命令，而且它们用于处理非数值作业，所以在这里，函数和命令之间的语法区别不作重点介绍。

2.1.5 在线帮助

MATLAB提供的内置函数帮助手册不仅有印刷版本，而且有HTML和PDF格式的电子文档[⊖]。但是，这些都不如使用MATLAB的在线帮助方便。我们可以通过在线帮助知道某些内

⊖ 在网站www.adobe.com上有了Adobe Portable Document Format的相关信息。

置函数的使用方法。

要获得某个函数*functionName*的在线帮助，需要在命令提示符后输入：

```
help functionName
```

或

```
helpwin('functionName')
```

这两种方法的区别体现在显示信息的方式不同："help *functionName*"在命令窗口中显示帮助信息，"helpwin('*functionName*')"在新打开的帮助浏览器(*help browser*)中显示帮助信息。

比如，当你想知道log函数是自然对数还是以10为底的对数时，可以不用查找手册，只需输入：

```
>> help log
```

这时，MATLAB将会显示：

```
LOG Natural logarithm.
 LOG(X) is the natural logarithm of the elements of X.
  Complex results are produced if X is not positive.

 See also LOG10, EXP, LOGM.
```

23

窗口的输出简明地显示了用户所要求的信息，而且显示了相关函数log10、exp、logm。

lookfor命令可用来查找某个特殊主题的相关函数。当函数名未知时，用lookfor查找函数可以知道MATLAB是否提供所需操作的相关函数。输入：

```
lookfor searchString
```

会对MATLAB中所有的函数进行搜索。如果*searchString*是某个函数名的一部分或者函数第一行注释的一部分，那么命令窗口会显示第一行注释行。比如，我们想知道MATLAB内置函数能否计算双曲余弦值。用lookfor查找"hyperbolic"或"cosine"时，MATLAB会返回相关函数名。例如，

```
>> lookfor cosine
ACOS    Inverse cosine.
ACOSH   Inverse hyperbolic cosine.
COS     Cosine.
COSH    Hyperbolic cosine.
```

该显示结果将列出函数名和对应功能的简短描述。用lookfor找到函数名后，通过help能够获得函数更详细的信息。例如，要查找cosh函数的详细信息，用户只要输入下列命令即可：

```
>> help cosh
```

2.2 矩阵与向量

前面例子中的变量都是标量。事实上，MATLAB中的所有变量都是数组(*array*)。通过变量名引用数组，通过下标(*index*)引用数组中的元素。在MATLAB 5及后续版本中，每个数组可以创建多个索引，在本书的应用中，每个数组只能提供一个或者两个索引，包括数值或者字符（在2.3.2节中介绍字符型数组，字符型数组也称为字符串(*string*)）。

矩阵是一个二维数组，它遵循的规则与第7章介绍的线性代数中矩阵的构造规则相同。本书的重点在于介绍如何定义和使用MATLAB中的矩阵变量来进行简单的计算。在MATLAB中，

24

标量可以看作是具有一行一列的矩阵，向量被看作是只有一行或一列的矩阵。在命令窗口中输入数学表达式后，命令解释器根据线性代数计算规则解析并计算表达式的值。对标量的操作同样遵循线性代数计算规则，如

```
>> a = 2; b = 3;
>> c = a*b
c =
     6
```

用方括号[]把向量或矩阵的元素括起来。下面的例子创建了包含3个元素的行向量：

```
>> v = [7 3 9]
v =
     7   3   9
```

分号运算符用于隔开每行。每行的元素之间用空格运算符或逗号运算符隔开。下面的例子创建了包含3个元素的列向量。

```
>> w = [2; 6; 1]
w =
     2
     6
     1
```

3行3列的矩阵如下所示：

```
>> A = [1 2 3; 5 7 11; 13 17 19]
A =
     1     2     3
     5     7    11
    13    17    19
```

当手工输入矩阵的值时，可用回车符将两行隔开，如：

```
>> B = [ 11    12    13      % press "return" after "13" is typed
         14    15    16
         17    18    19 ]
B =
    11    12    13
    14    15    16
    17    18    19
```

当矩阵的维数没有明确地定义时[⊖]，MATLAB会自动跟踪并计算行与列的数量以获得矩阵的维数。

2.2.1 创建矩阵

读者可以使用以下任意一种方法来创建MATLAB的矩阵变量：

- 手工输入
- 用数学表达式得到矩阵
- 使用内置函数，函数的返回值是矩阵
- 用户编写函数，函数的返回值是矩阵

⊖ 为提高效率可以预先分配矩阵，但这没有必要。

25

- 从磁盘文件中导入矩阵数据

前面的例子讲述了手工输入矩阵的方法。输入的时候，要区别行向量和列向量。线性代数中行向量和列向量是两个不同的向量，MATLAB命令解释器无法识别那些不符合线性代数规则的表达式。用转置操作符(*transpsoe operator*)，能把行向量变换成列向量，也能把列向量变换成行向量。比如说向量$\boldsymbol{v}$，它的转置向量是$\boldsymbol{v}^T$。例如：

$$\boldsymbol{v}=[2\ \ 4\ \ 1\ \ 7]\ \text{且}\ \ \boldsymbol{v}^T=\begin{bmatrix}2\\4\\1\\7\end{bmatrix}$$

MATLAB中转置操作符是单引号“'”；转置操作就是在矩阵或向量后面加个单引号。在MATLAB中创建向量v和它的转置向量v^T如下所示

26

```
>> v = [2 4 1 7]
v =
     2   4   1   7
>> v'
ans =
      2
      4
      1
      7
```

同样，用“'”号也能够转置矩阵。

```
>> A = [1 2 3; 4 5 6; 7 8 9 ];
>> A'
ans =
      1    4    7
      2    5    8
      3    6    9
```

由于小小的单引号不容易看到，所以遗漏或多余的转置操作符导致的错误也常常很难发现。

有几个内置函数可用于创建矩阵和向量，在表2-2中列出了其中一些内置函数。函数eye能够带1个或2个变量创建一个对角线元素全为1的对角矩阵。当eye函数只带1个输入变量时，这个变量是用户指定的矩阵维数，系统创建一个单位方阵。比如：

```
>> C = eye(4)
C =
      1      0      0      0
      0      1      0      0
      0      0      1      0
      0      0      0      1
```

表2-2　用来创建矩阵的一些内置函数

函　数	返 回 值
diag	对角矩阵或对角矩阵中对角线上元素值的索引
eye	单位矩阵
ones	全1矩阵
rand	随机矩阵

（续）

函 数	返 回 值
zeros	全0矩阵
length	向量中元素的个数
linspace	线性空间中的元素构成的行向量
logspace	对数空间中的元素构成的行向量
size	矩阵中的行数与列数

27

eye函数带两个参数时，第一个参数表示所要创建矩阵的行数，第二个参数表示所要创建矩阵的列数。对角矩阵是指在下标$i=j$的位置填1，其他位置填0，其中，i是行下标，j是列下标：

```
>> D = eye(3,5)
D =
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
```

函数diag既可以用于创建对角矩阵，又可以引用对角线上的元素。创建对角线元素值确定的矩阵，要求输入变量为一个向量，读者可以先明确地定义这个向量，然后使用向量名作为输入变量。例如：

```
>> v = [1 2 3];
>> A = diag(v)
A =
     1     0     0
     0     2     0
     0     0     3
```

读者也可以在输入参数中定义向量，例如：

```
>> B = diag([1 2 1 2])
B =
     1     0     0     0
     0     2     0     0
     0     0     1     0
     0     0     0     2
```

要引用已知矩阵对角线上元素的值，同样可以使用diag函数，此时的输入变量为一个矩阵名而不是向量名：

```
>> w = diag(B)
w =
     1
     2
     1
     2
```

和MATLAB中许多函数一样，diag函数的重要特性是要求用户输入变量然后系统作出响应。对于MATLAB的初学者，容易混淆同一个函数能够执行完全不同的（但它们之间是相关的）任务。MATLAB初学者掌握内置函数的这一特性将会有极大的帮助，会减少需要记忆的函数。

28

ones和zeros函数类似：ones函数创建的矩阵中所有元素的值为1，而zeros函数创建

的矩阵中所有元素的值为0。这两个函数都带两个输入变量，第一个变量表示矩阵的行数，第二个变量表示矩阵的列数，如：

```
>> D = ones(3,3)
D =
     1     1     1
     1     1     1
     1     1     1
```

矩阵不必是方阵：

```
>> E = ones(2,4)
E =
     1     1     1     1
     1     1     1     1
```

实际上，函数ones和zeros也可用于创建行向量和列向量，如：

```
>> s = ones(1,4)
s =
     1     1     1     1

>> t = zeros(3,1)
t =
     0
     0
     0
```

函数linspace用于创建行向量，向量中元素值之间的差值相同，它有两种表示形式：

```
linspace(startValue,endValue)
```

和

```
linspace(startValue,endValue,numPoints)
```

*startValue*和*endValue*分别是序列的起始元素的值和终止元素的值，*numPoints*是所要创建的向量的元素个数。例如：

```
>> u = linspace(0.0,0.25,5)
u =
          0    0.0625    0.1250    0.1875    0.2500
```

29

在上面的两个参数的表示形式中*numPoints*的默认值为100。

在这种标准形式下，函数linspace创建行向量。要创建列向量，只需加上转置操作符，如：

```
>> v = linspace(0,9,4)'
v =
     0
     3
     6
     9
```

函数logspace与函数linspace类似，只是logspace函数创建的矩阵中元素间距相等，且为对数步长。语句：

```
logspace(startValue,endValue,numPoints)
```

创建的元素值从$10^{startValue}$到$10^{endValue}$，元素的个数为*numPoints*个，如下所示：

```
>> w = logspace(1,4,4)
w =
```

```
             10          100         1000         10000
```

当省略第三个参数时，系统会在10[illegible]和10[illegible]之间自动创建50个元素，即*numPoints*的默认值为50。

例2.1 用表格的形式表示三角函数值

用下列MATLAB语句可创建一个向量，向量的元素是三角函数值。这些向量构成矩阵的列向量，以便这些元素值按表格的形式显示：

```
>> x = linspace(0,2*pi,6);
>> s = sin(x);
>> c = cos(x);
>> t = tan(x);
>> [x' s' c' t']
ans =
         0         0    1.0000         0
    1.2566    0.9511    0.3090    3.0777
    2.5133    0.5878   -0.8090   -0.7265
    3.7699   -0.5878   -0.8090    0.7265
    5.0265   -0.9511    0.3090   -3.0777
    6.2832         0    1.0000   -0.0000
```

矩阵[x's'c't']中的列向量是向量x、s、c和t的转置向量。如果遗漏转置符将会发生什么情况？能否使矩阵的第一行为向量x的值，第二行为向量s的值，第三行为向量c的值，第四行为向量t的值？请大家动脑筋思考一下！

[30]

2.2.2 矩阵元素的下标符号

下面通过类似Fortran中下标的使用来引用矩阵中的元素。对行向量或列向量，如果用v表示矩阵的行向量或列向量，那么，用$v(k)$引用向量中的第k个元素。当$\boldsymbol{A}$是矩阵时，用$\boldsymbol{A}(m,n)$来引用矩阵第m行、n列元素，如

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(3,2)
ans =
     8
```

当引用矩阵元素的下标超出矩阵的维数时，系统会显示出错：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(1,4)
??? Index exceeds matrix dimensions.
```

也可以使用下标符号给矩阵中的元素赋值，如：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(1,1) = 2
A =
     2     2     3
     4     5     6
     7     8     9
```

给下标超出矩阵维数的矩阵元素赋值是合法的。这时，MATLAB系统会扩展当前矩阵的维数来容纳新的元素，如：

```
>> D = eye(3,5)
D =
```

```
      1     0     0     0     0
      0     1     0     0     0
      0     0     1     0     0

>> D(4,2) = 2
D =
      1     0     0     0     0
      0     1     0     0     0
      0     0     1     0     0
      0     2     0     0     0
```

31

矩阵根据需要自动增长的特性简化了解决某些线性代数问题的冗余度和复杂度。但是，像MATLAB的其他特性一样，这一特点也会使MATLAB隐藏那些致命性的错误（参看练习18）。

函数length和size用于求出向量或矩阵中的元素个数，这有利于对未知长度矩阵或变长矩阵进行操作，特别有利于在循环体中使用(参看3.4.5节和3.4.6节)。下面的例子是创建一个向量并用0替换其最后一个元素：

```
>> x = 0:5     % define an sample vector
x =
     0   1   2   3   4   5

>> n = length(x)
n =
     6

>> x(n) = 0
x =
     0     1     2     3     4     0
```

函数size返回两个值，语法如下所示：

```
[nrows,ncols] = size(matrix)
```

这里*nrow*和*ncols*分别表示矩阵(*matrix*)中的行数和列数。要求用方括号把这两个参数括起来并用逗号隔开。以下列出函数size的使用方法：

```
>> A = eye(3,5);
>> [nr,nc] = size(A)
nr =
     3
nc =
     5

>> B = zeros(size(A))
B =
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

第二个例子表明size函数的返回值可用于其他命令中。这种写法常用于初始化另一个大小相同的矩阵。在这个例子中，也可以这样定义：B= zeros(nr,nc)，这里通过[nr,nc]=size(A)分别给nr和nc赋值。

32

2.2.3 冒号运算符

冒号运算符是MATLAB中的一个重要符号。它可用于创建变量，或者和下标一起使用来

引用矩阵中的元素，冒号运算符虽然微不足道，但是它的作用不容忽视。

使用冒号运算符创建向量有两种形式：

```
vector = startValue:endValue
vector = startValue:increment:endValue
```

这里，startValue和endValue都是区间的端点值，increment是区间的步长，vector为结果向量。参照下面的示例：

```
>> s = 1:5
s =
     1     2     3     4     5

>> t = 0:0.1:0.5
t =
     0    0.1000    0.2000    0.3000    0.4000    0.5000
```

第二个例子表明startValue、increment、endValue不必是整数。和linspace一样，冒号运算符默认创建行向量。当要创建列向量时，将表达式用圆括号括起来并加上转置符号即可，如：

```
>> u = (1:5)'
u =
     1
     2
     3
     4
     5
```

由于转置符号的优先级高于冒号运算符，所以必须使用圆括号将所得行向量括起来[⊖]。例如： [33]

```
>> v = 1:5'
v =
     1     2     3     4     5
```

在这里的转置作用在标量数5上，所以最后仍然得到一个行向量。

冒号运算符也能作为通配符（wild card）用来引用行向量或列向量，如：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(:,1)
ans =
     1
     4
     7

>> A(2,:)
ans =
     4     5     6
```

使用冒号运算符作为下标能够同时引用矩阵中的多个元素，如：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(2:3,1)
ans =
     4
```

⊖ 优先级的规则在3.4.2节介绍

```
     7

>> A(1:2,2:3)
ans =
     2     3
     5     6
```

冒号运算符也能用于赋值操作中，下面的例子用来创建一个全1矩阵并给第一行赋新值:

```
>> B = ones(3,4);
>> B(1,:) = [2 4 6 8]
B =
     2     4     6     8
     1     1     1     1
     1     1     1     1
```

第二条语句通过使用冒号运算符能够缩短为:

34

```
>> B(1,:) = 2:2:8
```

当要获得矩阵的最后一个元素或倒数第二个元素时，可以用如下的方法:

```
>> x = ...             %  define the x vector
>> s = x(end);         %  last element in x
>> t = x(end-1);       %  second-to-last element in x, etc.
```

本文中，关键字end的值等于向量的长度，使用end就可以避免定义一个变量来存储向量的长度。例如:

```
>> x = rand(1,5)
x =
   0.7621    0.4565    0.0185    0.8214    0.4447

>> x(end)
ans =
   0.4447

>> x(end-1)
ans =
   0.8214
```

显然，对于矩阵，关键字end表示矩阵的最末行或最末列，如:

```
>> A = [1 2 3; 4 5 6; 7 8 9];   %  define a matrix
>> B = A(2:end,1:end-1)         %  extract lower left corner
B =
     4     5
     7     8
```

2.2.4 从向量或矩阵中删除元素

通过给矩阵中的元素赋空值的方法能够删除其单个元素或一组元素。当定义一个向量x后，使用x = []后，向量x中的所有元素将被清除。用下标能够清除所选定的元素，如:

```
>> x = 1:5;           %  Create a sample x vector
>> x(3) = []          %  and delete the third element
x =
     1     2     4     5
```

35

用冒号运算符能够删除一组元素，如:

```
>> x = 1:5;
>> x(1:3) = []                    % Delete the first 3 elements
x =
     4     5


>> x = 1:10;
>> x(1:2:9) = []                  % Delete odd elements
x =
     2     4     6     8    10


>> x = 1:10;
>> x(length(x)-3:length(x)) = []  % Delete last 4 elements
x =
     1     2     3     4     5     6
```

上面最后一个语句能够用end关键字简化成如下形式:

```
>> x = ...
>> x = (end-3,end) = []
```

对矩阵做删除操作时，至少要删除矩阵的某行或某列:

```
>> A = [1 2 3; 4 5 6; 7 8 9];      % Sample matrix
>> A(:,1) = []
A =
     2     3
     5     6
     8     9


>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(3,3) = []
??? Indexed empty matrix assignment is not allowed.
```

这里，为了保证矩阵仍是一个矩形结构，不能只删除第三行第三列的元素。

2.2.5 对矩阵的数学运算

根据线性代数对矩阵运算规则的规定，可以对矩阵进行加、减、乘等运算。例如，可以对两个长度相等的行向量进行加或减运算: [36]

```
>> u = [10 9 8];    % u and v are row vectors
>> v = [1 2 3];
>> u+v              % sum is element by element
ans =

    11    11    11

>> u-v              % difference is element by element
ans =

     9     7     5
```

乘法运算能够对矩阵与矩阵、矩阵与向量间进行操作，可以是内积或外积，具体进行何种操作根据操作数的不同而不同（这里只简单介绍矩阵-矩阵、矩阵-向量和内积的运算，在第7章将作更详细的介绍）。如果两个矩阵A和B能够相匹配，MATLAB会根据线性代数中的矩

阵运算规则计算$A*B$。下面的例子是计算矩阵与矩阵、矩阵与向量的积：

```
>> A = ones(2,3);     B = [1 2; 3 4; 5 6];
>> A*B
ans =
     9    12
     9    12


>> C = diag(1:3); D = ones(3,3);
>> C*D
ans =
     1     1     1
     2     2     2
     3     3     3


>> x = [1; 0; 1];
>> A*x
ans =
     2
     2

>> C*x
ans =
     1
     0
     3
```

37

两个向量内积，也称为点积，其结果是一个标量。在线性代数中，内积操作是指行向量乘以列向量。例如，包含4个元素的行向量u与包含4个元素的列向量v的内积为：

$$uv = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = u_1v_1 + u_2v_2 + u_3v_3 + u_4v_4$$

如果向量v和w都是列向量，那么v和w内积之前需要先对向量v转置进行转置操作。假设向量v和w都有4个元素，得到

$$v^Tw = \begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = u_1w_1 + u_2w_2 + u_3w_3 + u_4w_4$$

用操作符*能够实现内积运算，如：

```
>> u = [10 9 8 6];  v = [1; 2; 3; 4];
>> u*v
ans =
    76

>> w = [1; 0; 1; -1]
w =
```

```
      1
      0
      1
     -1
>> v'*w          % notice the transpose on v
ans =
      0
```

内置函数dot也能够完成向量内积运算。它需要两个输入变量，这两个变量可以是行向量，也可以是列向量，要求它们的元素个数相等。如：

```
>> dot(u,v)
ans =
    76
>> dot(v,w)
ans =
     0
```

38

函数dot在计算之前会进行一致性检查和变维，所以它的运算效率不如*的运算效率高。

向量化和数组运算符 MATLAB中所有内置函数都已向量化，也就是说，给定输入向量后，函数对向量中的所有元素进行操作。这种操作有严格的语法要求。最重要的是，用向量化操作研究矩阵的元素时，比使用loop[⊖]对所有元素操作更能提高系统的计算效率。

以cos函数为例：

```
>> x = 0:pi/4:pi
x =
         0    0.7854    1.5708    2.3562    3.1416

>> y = cos(x)
y =
    1.0000    0.7071         0   -0.7071   -1.0000
```

在计算表达式y=cos(x)值的时候，MATLAB的命令解释器检测到x为一个向量。这极大地方便了MATLAB中的计算，使表达式更加优美。用Fortran语言实现上面的计算过程如下所示：

```
      real x(5),y(5)
      pi = 3.14159624
      dx = pi/4.0
      do 10 i=1,5
         x(i) = (i-1)*dx
         y(i) = cos(x(i))
10    continue
```

为了支持向量化，MATLAB中定义了一个新的算符，叫做数组运算符，它对两个行数、列数相等的矩阵（或向量）的相应元素进行运算。计算结果是产生和这两个矩阵有相同行数和列数的新矩阵。这在线性代数中没有直接与之等同的定义。使用数组运算符时，系统把矩阵看成一个数据结构，而不是一个数学实体（mathematical entity）。

39

数组运算符是用句点和传统运算符*、/和^的组合。点乘写作.*，点除写作./，如：

```
>> w = [1 2 3];    x = [4 5 6];   % Two row vectors
>> y = w.*x                       % Element-by-element multiplication
```

⊖ 在第3章的3.4.5节和3.4.6节中介绍循环。

```
y =
     4    10    18

>> z = w./x                          % Element-by-element division
z =


    0.2500    0.4000    0.5000
```

数组操作既适用于向量，又适用于矩阵：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> B = [9 8 7; 6 5 4; 3 2 1];
>> A.*B
ans =
     9    16    21
    24    25    24
    21    16     9
```

注意这里A.*B不同于A*B。

数组求幂运算符.^是表示对矩阵中每个元素进行求幂操作，如：

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A.^2
ans =
     1     4     9
    16    25    36
    49    64    81

>> A.^(1/2)
ans =
    1.0000    1.4142    1.7321
    2.0000    2.2361    2.4495
    2.6458    2.8284    3.0000
```

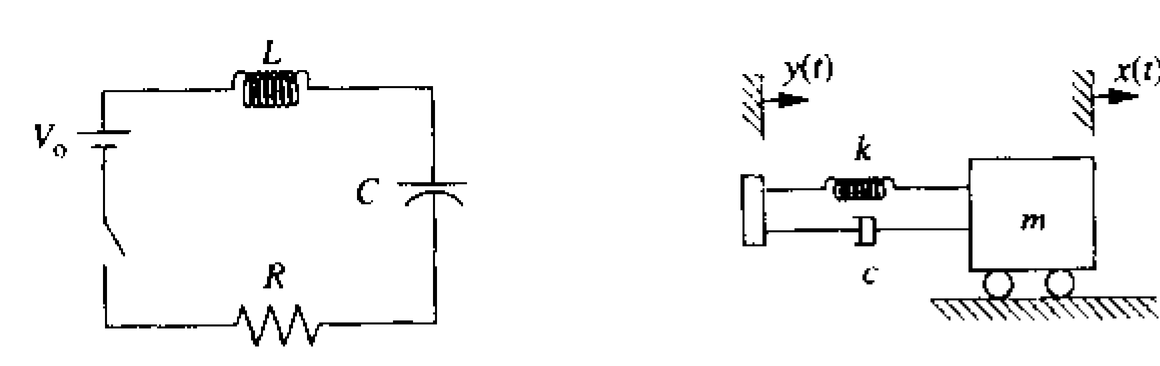
数组运算符的语法限制要求印刷时特别要注意小符号的使用，比如说一个复杂公式中的句点符号就容易被忽略，但是它在计算中可能会起很大的作用。MATLAB虽然能够检测出语法错误，但是像下面列举的这类错误却无法检测出来。比如，A.^2和A^2都是合法的，但是运算所得的结果则完全不同。

[40]

在线性代数中，矩阵加和矩阵减都是对元素进行操作，没有必要使用特殊的数组运算符。

例2.2 有阻尼的二阶系统的响应

下图左边描述一个二阶电路系统，右边描述一个相似的二阶机械系统：



欠阻尼二阶系统对阶跃输入的响应如下所示：

$$x(t)=u_0\left[1-e^{-\zeta\omega_n t}(\cos\omega_d t-\gamma\sin\omega_d t)\right]$$

其中u_0是阶跃函数的阶跃值。

$$\omega_d=\omega_n\sqrt{1-\zeta^2}, r=\frac{\zeta}{1-\zeta^2}$$

$$\omega_n=\sqrt{\frac{1}{LC}}\ （电子的）或\ \omega_n=\sqrt{\frac{k}{m}}\ （机械的）$$

并且

$$\zeta=\sqrt{\frac{R}{2L}}\ （电子的）或\ \zeta=\frac{c}{2\sqrt{km}}\ （机械的）$$

当$\omega_n=104$，$\zeta=0.3$，$u_0=1$，$0\leqslant t\leqslant 0.2$时，计算对应公式所得结果。首先给变量赋值[⊖]并创建区间从0到0.2的时间向量：

41

```
>> omegan = 104;   zeta = 0.3;   u0 = 1;
>> omegad = omegan*sqrt(1-zeta^2);
>> gam = zeta/sqrt(1-zeta^2);
>> t = linspace(0,0.2);
```

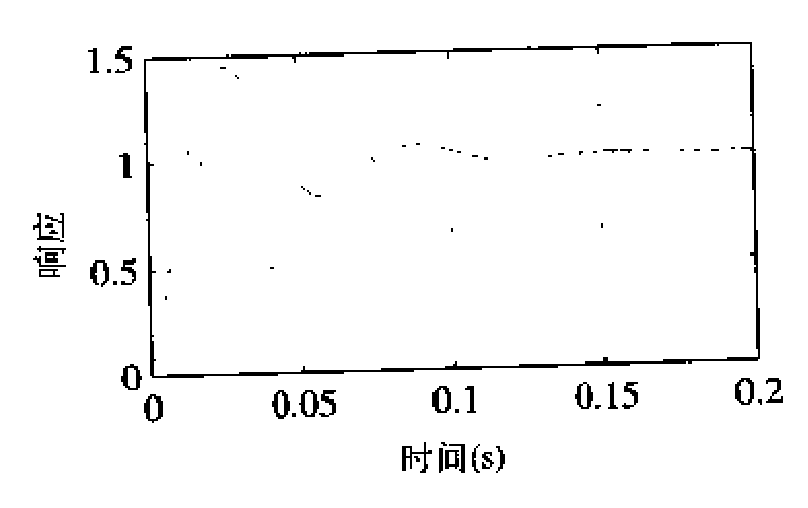


图2-2 当$\omega_n=104$，$\zeta=0.3$，$u_0=1$时，欠阻尼二阶系统的响应

下一步计算响应公式的值。在C语言或Fortran语言中，需要通过循环实现对与每个t值对应x的计算。在MATLAB中，只用一条语句就可以实现：

```
>>x=u0*(1-exp(-zeta*omegan*t).*(cos(omegad*t)-gam*sin(omegad*t)));
```

使用数组运算符.*是因为表达式`exp(-zeta*omegan*t)`、`cos(omegad*t) - gam*sin(omegad*t)`都生成了包含100个元素的行向量。将它们点积以后的值赋给向量x。大家思考一下，如果用*代替.*将会是什么结果？

图2-2是利用MATLAB创建的$x-t$曲线图。创建这个图的命令我们将在2.5节中介绍。

2.2.6 矩阵变维

有的时候，存储在矩阵中的数据只需要变维，而不需要进行数学转换。比如，为了使计算简便，有时需要把矩阵转换成向量，在形式上就是从一个方矩阵变成一个长矩阵。函数reshape用来完成这一操作。在某些情况下，使用冒号运算符可能更方便些。

函数reshape带有3个输入变量，比如：

```
outputMatrix = reshape(inputMatrix,newRows,newColumns)
```

这里*inputMatrix*是输入矩阵，*newRows*和*newColumns*分别是*inputMatrix*矩阵

⊖ 为了区别内置函数gamma，这里用gam代替gamma。

42 改造后输出的新矩阵的行数和列数，它们的乘积必须和输入矩阵的元素个数相等，且最后一列必须填满数据。下面是使用函数reshape的例子：

```
>> A = [1 5 9; 2 6 10; 3 7 11; 4 8 12]
A =
     1     5     9
     2     6    10
     3     7    11
     4     8    12

>> B = reshape(A,2,6)
B =
     1     3     5     7     9    11
     2     4     6     8    10    12

>> s = reshape(A,1,12)
s =
     1   2   3   4   5   6   7   8   9   10   11   12
```

矩阵A中元素的选取是为了使读者能更明确地认识reshape函数如何将矩阵中的元素按列变维。

行向量和列向量是矩阵的特例，因此reshape函数能将向量变维为矩阵，如：

```
>> t = 1:6;
>> C = reshape(t,2,3)
C =
     1     3     5
     2     4     6
```

也能适当地改变矩阵的形状（在不拷贝到另外一个矩阵的情况下），如：

```
>> D = [ 1 2 3; 1 2 3; 1 2 3 ]
D =
     1     2     3
     1     2     3
     1     2     3

>> D = reshape(D,1,9)
D =
     1     1     1     2     2     2     3     3     3
```

冒号算符能够把各种矩阵转换成列向量。它有特殊的语法：

43 *columnVector = matrix(:)*

输出结果总是列向量，而且等号右边的矩阵的行数和列数任意，如：

```
>> E = [1 4; 2 5; 3 6]
E =
     1     4
     2     5
     3     6

>> v = E(:)
v =
     1
     2
     3
```

```
     4
     5
     6
```

当需要生成行向量时，只要加上转置运算符即可，如：

```
>> E = [1 4; 2 5; 3 6];
>> w = E(:)'
w =
     1     2     3     4     5     6
```

和函数reshape一样，冒号运算符也能适当地对向量进行操作，如：

```
>> y = 1:5;
>> y = y(:)
y =
     1
     2
     3
     4
     5
```

在要求输入变量为一个行向量或列向量时，使用冒号运算符进行变维操作将很方便。行向量与列向量的转换只需要增加一个转置符号，反之，也只需要一个转置符号。

变维操作充分利用了MATLAB中内置向量化的特性，这比将数据一个一个拷贝到所求矩阵中更快捷有效。

44

2.3 其他变量类型

在MATLAB的计算中，经常要用到数值型变量或字符型变量。字符型变量主要用在图形的标注和用户定义的函数名中。数值型变量可以是实数或复数。本节介绍MATLAB中支持复数和字符串操作的内置特性，包括对多项式操作的函数的介绍。

用户在MATLAB中定义的变量是包含好几段不同数据的对象（*object*）。对象中一部分数据包含数值数据或字符数据，我们认为它们存储在变量中；另一部分数据包含对象的长度（如矩阵中的行数和列数）。而且，每个对象包含由MATLAB使用的内部属性，其中有一个属性是一个标识，用于表明目标变量中是否存储了虚数。当然，用户不用考虑这些内部属性，它由MATLAB系统本身使用。

2.3.1 复数

复数计算已经完全集成到MATLAB中，MATLAB将所有变量都看成复数，而且+、-、*、/操作时，系统将自动对标量、向量和矩阵元素的实部和虚部进行计算。单位虚数被系统预定义为$i = j = \sqrt{-1}$。应用i和j表示对单位虚数的引用，如：

```
>> x = 1 + 2*i        %  or, if you prefer,  x = 1 + 2*j
x =
   1.0000 + 2.0000i

>> y = 1 - 2*i
y =
   1.0000 - 2.0000i

>> z = x*y
```

```
z =
     5
```

根据上面几条语句可以看出，虽然在输入虚数时j可以代替i,但是在MATLAB中，通常使用i表示单位虚数，如:

```
>> x = 1 + 2*j
x =
   1.0000 + 2.0000i
```

45

表2-3　对复数操作的内置函数

函　数	操　作
abs	计算复数的模[如，abs(z)=sprt(real(z)^2+imag(z)^2)]
angle	欧拉（Euler）表示法中的复角
conj	共轭复数
imag	得到复数的虚部
real	得到复数的实部

给复数赋值时，常数与i或j的乘积可以用*号也可以不用。也就是:

```
>> x = 1 + 2*i
```

和

```
>> x = 1 + 2i
```

是等价的。只要单位虚数值i和j处于表达式的尾部，就可以省略*号。但是，表达式x = 1+i2是错误的。同时，i或j不能和变量名一起使用，否则，会引起混乱。如:

```
>> w = 2;
>> x = 1 + wi
??? Undefined function or variable wi.
```

注意到i和j都作为普通的MATLAB变量，它被预定义为 $\sqrt{-1}$ ，当在程序中给它们重新赋值后，前面给出语句的输出结果就不同了，如下列计算:

```
>> i = 5;    t = 8;    u = sqrt(t-i)
u =
    1.7321
```

赋值语句i = 5把变量i以前的值覆盖了，所以t - i结果为3而不是8 - $\sqrt{-1}$ 。为了不致于混淆，用户应该知道下面的常识性规则:

> **使用复数时**
> **保留i和j**
> **它们的值都为 $\sqrt{-1}$**

46

在其他语言中，i和j常被用于作为数组下标，在MATLAB中也可以这样使用，但是要给它们赋初值，如:

```
>> A = [1 2; 3 4];  i = 2;  A(i,i) = 1
A =
     1     2
     3     1
```

只要复数的虚部有意义，复数就能用于任何计算中。表2-3列出了对复数操作的内置函数，函数exp可用于复数的欧拉表示法中，如$z=\zeta e^{i\theta}$，这里的ζ是复数的模，θ是复数在极坐标中的复角。图2-3画出了复数的欧拉表示法及其在复平面中对应的笛卡儿坐标。例如，当$\zeta=5$，$\theta=\pi/3$时：

```
>> zeta = 5;  theta = pi/3;  z = zeta*exp(i*theta)
z =
   2.5000 + 4.3301i

>> abs(z)
ans =
     5

>> angle(z)*180/pi
ans =
   60.0000
```


图2-3 复数的欧拉表示法

这里，z的输入也可以用笛卡儿坐标表示法代替欧拉表示法。函数real、imag、conj、abs和angle简化了欧拉表示法和笛卡儿坐标表示法之间的转化。

47

乘法和除法操作自动完成复数实部和虚部的运算，如：

```
>> s = 3*exp(i*pi/3);  t = 3*exp(i*2*pi/3);
>> s-t
ans =
    3.0000

>> s+t
ans =
   0.0000 + 5.1962i

>> s*t
ans =
    -9.0000 + 0.0000i

>> s/t
ans =
   0.5000 - 0.8660i
```

上面的计算也适合对复数矩阵中的元素所进行的运算，如：

```
>> A = [1 2; 3 4] + i*[1 2; 3 4]
A =
   1.0000 + 1.0000i 2.0000 + 2.0000i
   3.0000 + 3.0000i 4.0000 + 4.0000i

>> B = abs(A)
B =
    1.4142  2.8284
    4.2426  5.6569

>> C = A*A'
C =
    10     22
    22     50

>> D = A'*A
```

```
D =
    20    28
    28    40
```

2.3.2 字符串

字符串也是矩阵，矩阵中的元素是字符。一般的赋值和创建规则都可用于字符串操作。[48] 创建字符串的简单方法是，给字符串赋值，等号的右边用来计算字符串的值并将其赋给等号左边的变量。字符串常量用单引号括起来，如：

```
>> first = 'John';
>> last  = 'Coltrane';
>> name  = [first,' ',last]
name =
John Coltrane
```

前两行定义了两个字符串变量。第三行是由其他三个字符串变量创建一个新的字符串变量，这种赋值操作是连接字符串的最直接的方法。也可以用冒号运算符引用某个字符串的子串，如：

```
>> sentence = 'The red ball is on the table';
>> subject = sentence(9:12)
subject =
ball
```

字符串通常是行向量，但在有的情况下，也可以是列向量，如：

```
>> color = sentence(5:8)'
color =
r
e
d
```

由于转置符号和字符串定界符一样都是单引号，所以在直接给列向量赋值时，需要给字符串加上圆括号，如：

```
>> newColor = ('blue')'
newColor =
b
l
u
e
```

也可以创建字符串矩阵，如：

```
>> fruit = ['apples ';'oranges';'kiwis  ']    % notice extra blanks
fruit =
apples
oranges
kiwis
```

和数值矩阵相同，字符串矩阵每行的长度必须相等。创建fruit变量时，为了使apples [49] 和kiwis的长度等于oranges的长度，给它们后面都加上了空格符。函数str2mat能够自动把字符串矩阵中的每行进行填充，使它们的长度相等，此函数的每个输入参数（字符串）分别放在不同的行。前面的语句能够用下面的语句来替换：

```
>> fruit = str2mat('apples','oranges','kiwis')
```

```
fruit =
apples
oranges
kiwis
```

创建图形（参照2.5.2节）或打印输出时（参照3.3.2节），有时要利用数值数据创建字符串。它可以用num2str函数或sprintf函数来实现。函数num2str的语法比较简单，而sprintf函数语法稍复杂一些，用户可以用它实现数值-字符串的精确转换。函数num2str的一个典型的用途就是将小的数字矩阵变成一个大的字符矩阵，如：

```
>> root3 = ['The square root of 3 is ',num2str(sqrt(3))]
root3 =
The square root of 3 is 1.732
```

注意到num2str(sqrt(3))的返回值是一个字符串值，所以没有必要加上引号。为了更加精确，函数num2str提供了第二个输入参数，如下所示：

```
>> root3 = ['The square root of 3 is ',num2str(sqrt(3),10)]
root3 =
The square root of 3 is 1.732050808
```

函数sprintf可以指定结果精确到的位数，如：

```
>> root3 = sprintf('The square root of 3 is %9.7f',(sqrt(3)))
root3 =
The square root of 3 is 1.7320508
```

输出结果也可以用指数形式表示，如：

```
>> root3 = sprintf('The square root of 3 is %11.5e',(sqrt(3)))
root3 =
The square root of 3 is 1.73205e+00
```


函数sprintf的格式部分同函数fprintf（在3.3.2节介绍）的格式部分一样，它们都由相同的小矩阵构成（想知道更多信息的读者请参照文献[73]*Using MATLAB*。

2.3.3 多项式

MATLAB提供了许多对多项式操作的内置函数。多项式是由实系数或复系数构成的向量，它并不是一种数据类型。对多项式的计算实际上是对这些系数的计算。普通的算术运算符（+、-、*）或数组运算符（.*、./、.^）等都能对多项式的系数进行操作，但计算结果要根据线性代数运算规则和MATLAB操作符优先级而定。

可以用系数向量表示按降幂顺序列出的n阶多项式，如：

$$P_n(x)=c_1x^n+c_2x^{n-1}+\cdots+c_nx+c_{n+1} \quad (2\text{-}1)$$

例如，多项式$x^3-2x+12$可用向量[1 0 -2 12]明确地表示。给定系数向量，函数polyval能够根据x的值计算最后结果，要计算当x=1.5时，多项式$x^3-2x+12$的结果，可以输入：

```
>> c = [1  0  -2  12];
>> polyval(c,1.5)
ans =
   12.3750
```

可以把上例中的两条语句结合起来使用（如，polyval([1 0 -2 12],1.5)）。如果输入

x为一个向量，那么函数polyval会针对x中的每个x_i值计算多项式的值并输出到一个向量中，如:

```
>> c = [1  0  -2  12];
>> x = 1:5;
>> polyval(c,x)
ans =
    11     16     33     68    127
```

表2-4列出了MATLAB第5版中对多项式操作的内置函数。

表2-4　对多项式操作的内置函数。这些函数的输入和输出都默认为按降序排列的系数向量

函　数	描　述
conv	求两个多项式的乘积多项式
deconv	求两个多项式的商多项式
poly	创建有特定根的多项式
polyder	求多项式的微分得到的多项式
polyval	求多项式$P_n(x)$在特定值x处的值
polyvalm	求方阵A的多项式表达式的值，方阵A的幂可按矩阵-矩阵乘积计算
polyfit	计算n阶多项式$y = P_n(x)$的系数，这个多项式是对数据集（x,y）的最小二乘曲线拟合（参照第9章）
residue	计算两个多项式相除以后的部分分式展开式；或给定余数、阶数和系数的向量，求出多项式相除后对应的系数比
roots	计算n阶多项式的n个根（参照第6章）

51

2.4　管理交互环境

在MATLAB交互环境中，有时候需要将那些已经定义的变量保存到磁盘中以便继续使用，或者使用那些已被定义而且保存了的变量，或者从内存中清除所有变量或某些变量，所有这些操作都要在MATLAB工作区（*workspace*）完成。

2.4.1　MATLAB工作区

当用户定义一个变量后，MATLAB存储管理器给数据变量和描述这一变量的各项属性分配一块内存（*random access memory*, RAM）。一旦变量定义以后，它就是MATLAB工作区的一部分，因此，MATLAB工作区也可以看成是变量列表。启动MATLAB并输入下面的命令，用户能够看到MATLAB是如何定义变量的:

```
>> clear
>> a = 5;  b = 2;  c = 1;
>> d(1) = sqrt(b^2 - 4*a*c);    d(2) = -d(1);
>> who
Your variables are:

a          b          c          d
```

who命令列出工作区中定义的变量和这些变量用到的内存。whos命令（who加“size”）提供更详细的信息:

52

```
>> whos

  Name        Size           Bytes  Class
```

```
a          1x1                8  double array
b          1x1                8  double array
c          1x1                8  double array
d          1x2               32  double array (complex)

Grand total is 5 elements using 56 bytes
```

该表列出了每一变量的矩阵维数（“1×1”为标量），矩阵变量总共占用的字节数和变量的“类型”。类型用于定义变量的信息在内存中如何存储和访问，以及允许对变量所进行的操作。内置类型是double、sparse、struct、cell和char。同时MATLAB也允许用户自定义变量类型。double类型用于完成基本的数值计算；char类型用于字符串操作，主要用于格式化输出或给图形作注解。文献[73]*Using MATLAB*中介绍了其他变量类型，供读者参考。

随着继续定义更多的变量，它们都被添加到工作区中。当使用clear命令或终止MATLAB交互环境时，工作区才不会继续增大。当用户处理脚本文件（在3.1节介绍）时，跟踪工作区的变量是非常重要的。在使用较大的矩阵时，有必要清除工作区中不再使用的变量来为矩阵提供足够的内存空间。这时，用户使用whos命令列出变量的属性并得出占用内存最多的变量，然后使用clear命令有选择性地清除一些不必要的变量。例如，要清除上面例子中的变量a和d，可以输入如下命令：

```
>> clear a d
```

2.4.2 处理外部文件中的数据

我们能够保存MATLAB交互环境的结果，然后在另一个MATLAB的交互环境中重新加载，也可以输出到磁盘上供其他程序使用。实验获得的数据或由其他程序保存的数据都能够输入到MATLAB中作进一步分析，最简单也是最灵活的输入输出命令分别是save和load，而且只要简单的一步操作就能够完成。一些功能稍强的函数，虽然使用起来不难，但是要求用户详细说明数据和文件之间的转换，增加了用户的操作。

53

save和load命令　save命令用于将活动交互环境中的变量写入一个文件中。save命令的简单使用格式如下所示：

```
save fileName
save fileName variable1 variable2 ...
save fileName variable1 variable2 ...  -ascii
```

上面的命令中都要指定目的文件名。如果没有给出变量列表，save命令把工作区中所有变量写入目的文件中。前两种形式的命令创建二进制“mat”文件。mat格式的文件包含变量名、变量大小和存储在矩阵元素中的数据。如：

```
>> clear
>> x = 0:5;   y=5*x;
>> save xyfile
```

创建文件xyfile.mat用于保存变量x和y[⊖]。和save命令等价的函数是：

```
save('fileName','var1','var2',...)
```

这里*fileName*是用于保存变量*var1*和*var2*的mat格式文件的文件名。用save的函数

⊖ 如果不用clear清除所有变量，文件xyfile.mat中会包含除x，y之外工作区中的其他变量。

形式创建xyfile.mat文件的语句如下所示:

```
>> x = 0:5;    y=5*x;
>> save('xyfile','x','y')
```

注意：输入参数是'x'和'y'（也就是变量名）。

"-ascii"格式的save命令用于创建纯文本文件，它可以用文本编辑器打开。如:

```
>> x = 0:5;    y=5*x;
>> XY = [x' y'];
>> save xyvals.txt  XY  -ascii
```

54

创建的纯文本文件包含下面数据:

```
0.0000000e+00   0.0000000e+00
1.0000000e+00   5.0000000e+00
2.0000000e+00   1.0000000e+01
3.0000000e+00   1.5000000e+01
4.0000000e+00   2.0000000e+01
5.0000000e+00   2.5000000e+01
```

在前面的例子中，矩阵XY需要显式地创建，这是因为在save命令中，MATLAB解释器不能够解析'字符和[]号。用ascii格式的save命令仅向文件写入矩阵中元素的值，向二进制mat格式的文件中写入的内容还包括变量名、变量大小和其他工作区信息。总之，当变量将来还要读入MATLAB时，使用非ascii格式的save命令；当变量要输出到其他程序时，如电子表格和文字处理程序，使用ascii格式的save命令。

load命令和save命令是成对的，load命令把数据读入MATLAB中。load命令有两种格式，如下所示:

```
load fileName
load fileName matrixVariable
```

load命令根据所读文件内容的不同而进行不同的操作。第一种格式中仅仅指定了文件名，它用于装载二进制mat文件，此时，mat文件中的变量在当前工作区被重新创建。第二种格式在*fileName*是一个ascii文件，且其中只包含对应于矩阵的数值列时使用，此时，创建变量*matrixVariable*并将文件中的数据读入此变量的对应列中。

load函数的语法如下:

```
D = load('fileName')
```

这里*fileName*可以是mat格式文件或纯文本文件。当它是纯文本文件时，它的扩展名可以为.dat或.txt，当它是mat格式文件时，由D=load('*fileName*')创建的变量D是MATLAB结构体。当*fileName*是一个纯文本文件时，变量D是一个包含文件中所有数据的矩阵。*fileName*也可以是文件的路径描述，如:

```
>> D = load('C:\myData\someFile.dat');        % DOS/Windows file system
```

55

或者

```
>> D = load('~/myData/someFile.dat');         % Unix file system
```

例2.3　从mat文件中装载矩阵

在下面的MATLAB交互环境:

```
>> clear
>> x = linspace(0,2*pi);    y = cos(x);    z = sin(x);
```

```
>> save trigvar
```

运行结果为在当前目录创建trigvar.mat文件并向其中写入数据。在命令行输入pwd察看当前目录、输入dir或ls察看当前目录中的文件名。

文件trigvar.mat包含变量x、y、z中的数据及其附带的信息。在接下来的某个时刻，可用下列语句重新向这一交互环境或新启动的MATLAB交互环境中装载这些数据，如：

```
>> clear             % clear all variables from workspace (not necessary)
>> whos              % Now the workspace is empty

>> load trigvar      % Read contents of trigvar.mat into the workspace
>> whos

  Name        Size            Bytes  Class

  x           1x100             800  double array
  y           1x100             800  double array
  z           1x100             800  double array

Grand total is 300 elements using 2400 bytes
```

尽管变量y和z都是由变量x得来，但在trigvar.mat文件中却没有明显地反映出来。事实上，变量间的数学关系在工作区文件中不作为保存的信息。

变量x、y和z也可以用load函数读入，如：

```
>> load trigvar
```

例2.4 从文本文件中装载数据

load命令的一个重要的应用是从纯文本文件中读入数据并进行分析和画图。此时，要求文件中的各列等长，且各列中的数据都是数字。不可以包含文本，如列标题。

在NMM工具箱中的data目录下的文件pdxTemP.dat包含俄勒冈州波特兰市的历史平均气温。这些数据按4列排列，每行对应一个月份。输入以下几条语句能够获得pdxTemp.dat文件的内容：　[56]

```
>> type pdxTemp.dat
```

如果出现"file not found"错误，说明工具箱NMM没有正确地安装到计算机中。察看NMM工具箱中的install.m文件和ReadMe.m文件。pdxTemp.dat文件中的第一列表示月份（从1月到12月），第二列到第四列分别是平均最高气温、平均最低气温和每月的平均气温。如，平均最高气温是本月中每天最高气温的平均值。这里，所有气温以华氏温度为单位。

下面几条语句把数据从pdxTemp.dat文件读入MATLAB变量中（右边是详细的注释）：

```
>> D = load('pdxTemp.dat')     % Data are stored in D matrix
>> month = D(:,1);             % Copy first column of pdxTemp to month
>> T = D(:,2:4)                % Copy columns 2 through 4 of pdxTemp to T
T =
   45.3600   33.8400   39.6000
   50.8700   35.9800   43.4300
   56.0500   38.5500   47.3000
   60.4900   41.3600   50.9200
   67.1700   46.9200   57.0500
   73.8200   52.8000   63.3100
   79.7200   56.4300   68.0700
```

```
   80.1400   56.7900   68.4700
   74.5400   51.8300   63.1800
   64.0800   44.9500   54.5200
   52.6600   39.5400   46.1000
   45.5900   34.7500   40.1700
```

将温度数据拷入矩阵T后，可以进行简单的计算。该年度的最高气温是：

```
>> Thigh_max = max(T(:,1))
Thigh_max =
   80.1400
```

该年度的最低气温是：

```
>> Tlow_min = min(T(:,2))
Tlow_min =
   33.8400
```

该年度的平均气温是：

```
>> Tave_ave = mean(T(:,3))     % Find the average of the monthly averages
Tave_ave =
   53.5100
```

57

前面的交互环境创建了下列变量：

```
>> whos
  Name             Size          Bytes  Class

  pdxTemp         12x4             384  double array
  T               12x3             288  double array
  Tave_ave         1x1               8  double array
  Thigh_max        1x1               8  double array
  Tlow_min         1x1               8  double array
  month           12x1              96  double array

Grand total is 99 elements using 792 bytes
```

注意：先把原始数据保存在pdxTemp矩阵中，把pdxTemp中的数据拷到变量month和T后，矩阵内容不变。

查找路径 当用户输入下列语句：

```
>> x = 2;
>> y = someFunction(x)
```

MATLAB在计算机中少量预定义的目录下查找叫做someFunction的m文件函数[⊖]。这些含有m文件和数据的预定义目录称为查找路径或简称为路径。若m文件（如，someFunction）不在查找路径中，表明MATLAB找不到请求函数，这时，MATLAB打印错误信息，如下所示：

```
??? Undefined function or variable 'someFunction'.
```

类似地，用户要求装载的文件不在查找路径的目录下时，也会出现错误信息，如：

```
>> load theFile
??? Error using ==> load
theFile.mat: File not found.
```

⊖ 第3章介绍m文件函数。现在，只要知道m文件函数用于定义类似内置函数的子程序即可。事实上，大多数内置函数都是m文件。

尽管查找路径是预先定义好的，但是用户能够改变预定义的查找路径。通常，查找路径能够被扩展到其他目录下以便用户查找。用户自定义查找路径使用户能够根据工程和应用的需要把文件分放在单独的目录中。 58

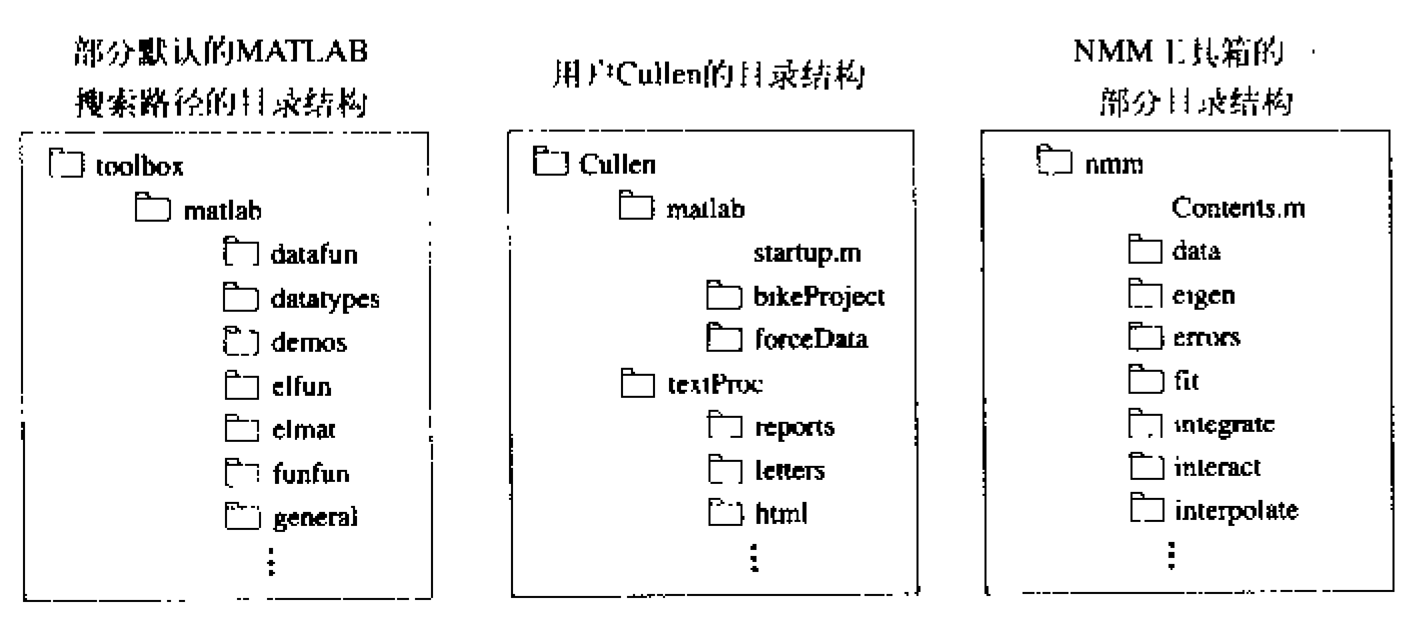


图2-4 MATLAB内置工具箱、用户Cullen和NMM工具箱的目录结构的部分列表

假如用户Cullen把文件存储在一台文件服务器的子目录下，那么这台文件服务器（或网络中的其他电脑）需装有MATLAB程序并且拥有NMM工具箱的副本。文件的存储路径如图2-4所示。3个目录树可能会存储在同一台电脑中，但是，用户Cullen只能修改其主目录下的文件。安装MATLAB后，查找路径包含了toolbox目录，为了使所有用户能够访问NMM工具箱，系统管理员需要把nmm目录树（图2-4最右边）添加到MATLAB搜索路径中，这可以通过修改MATLAB主目录下local目录中的pathdef.m文件来实现。或者，用户Cullen在目录matlab下安装NMM工具箱并把这一目录包含到搜索路径中。

如果用户Cullen想把自己的目录添加到查找路径中，可以使用内置path函数和addpath函数。但是，这些函数只会修改当前交互环境的路径。用户使用不同的操作系统，为MATLAB的交互环境添加查找目录的方法也各不相同。用户使用Unix操作系统时，需要修改其主目录中的matlab子目录下的startup.m文件。使用Windows或Macintosh操作系统时，可以使用路径浏览器（*path browser*）修改本次交互环境的默认路径。参考文献[73]*Using* MATLAB中针对不同的操作系统介绍了各种不同的路径设置方法。

低级输入/输出函数 函数load和save可以看成高级I/O函数。用户只要输入简单的语句，系统就能够自动向磁盘传送或接收来自磁盘的数据。这种简单的应用导致用户无法控制操作的细节。比如，命令load只能输入按mat格式存放的数据成按列排列的数值数据，当数据按列并带列标题[⊖]存放在纯文本文件中时，将不方便用户使用。为了对文件格式实现更多的控制，MATLAB提供了几个低级I/O函数。这里对这些函数作简单介绍，只是为了让读者知道如何读取带列标题的文本文件。使用这些函数，用户要根据不同的操作系统应用不同的方法。参照文献[73]*Using* MATLAB可以获得更多的信息。 59

函数fopen在文件和文件标识符(*file identifier*)之间建立连接。它的一般格式如下：

```
fid = fopen(fileName,permission)
```

*fileName*是要打开（或创建）的文件名，*permission*是字符串，表示许可的文件操作，*fid*是文件标识符。表2-5介绍了对文件最常用的文件许可操作。当打开文件操作成功时，

⊖ 用户可以用文件编辑器打开文件并删除列标题，但这样首先就改变了使用列标题的目的。

返回正整数给*fid*，当失败时，返回－1给*fid*。*fid*的值就能够应用于接下来的其他文件I/O操作。文件的输入输出操作完成后，用户最好使用fclose函数关闭与文件的连接以防文件意外损坏。fopen函数的典型用法如下所示：

```
>> fp = fopen('myFile.dat','rt');  % open text file myFile.dat for input
>> ...                             % read data from myFile.dat
>> fclose(fp)                      % close the file
```

例2.5列出了fopen函数的另一种用法。

表2-5　函数fopen的最常用的文件许可操作。输入'help fopen'可获得更多信息

许　可	文件操作
'r'	打开一个文件进行读操作
'w'	打开一个文件进行写操作。文件不存在时，先创建再写；已经存在时，将其原来内容覆盖(*overwrite*)
'a'	打开一个文件进行追加(*append*)操作。文件不存在时，先创建再追加；已经存在时，直接追加在文件末尾

60

注意：使用Windows和VMS操作系统的用户读写文本文件时，需要使用'rt'和'wt'许可。

用fopen打开文件后，用户可以使用fgetl和fscanf函数读这一文件。函数fgetl读取纯文本文件中的某一行，它的语法如下所示：

```
line = fgetl(fid)
```

其中*line*是包含文本行数据的字符型变量。不同的操作系统对纯文本文件的分行采用不同的约定，用特殊的字符序列来表示行的结尾。函数fgetl返回的*line*字符串中不包含行结尾字符序列。当应用要求返回行结尾字符序列（通常不需要）时，可以使用fgets函数（参考文献[73]*Using* MATLAB）。

函数fscanf将文件中的数据读到一个MATLAB变量中。C程序员尤其要注意MATLAB中的fscanf函数和C中的fscanf函数的不同点。MATLAB中fscanf函数的一般格式如下所示：

```
x = fscanf(fid,format)
x = fscanf(fid,format,size)
```

这里*fid*是由函数fopen返回的文件标识符，*format*是字符串，表示给变量*x*赋值所要求的转换格式（表3-2列出了格式转换字符串）。要读取数值数据，需要使用%f格式字符串，这将在例2.5中作介绍。

可选参数*size*表示所读数据的长度。若*size*为一个标量n，那么函数以format格式从文件中读取n个数据。若*size*为一个两元素的行向量[m　n]，函数fscanf的返回值是m行n列的矩阵。3.3.2节介绍如何使用fprintf函数向纯文本文件中写数据。

例2.5　从带列标题的纯文本文件中装载数据

通常情况下，数据文件包含正文标题和数值数据。只有删除了文件中的正文标题以后，用户才能用内置load命令读取文件中的数据。但是删除这些文件的正文标题并不实际，因为这些正文标题提供了数据的文档信息。为了能够读取正文标题和数值数据，用户可以使用低级I/O函数如fopen、fgetl和fscanf。

要读取文本和数值数据，用户必须提前知道文件的结构。特别地，用户必须知道文件有多少行以及相关数据的文本行的位置。函数fgetl用来读文本行，函数fscanf用来读数值

61

数据。一个fgetl函数只能一次读一行，而函数fscanf一次能够读多行。

例2.4中，用load命令将文件pdxTemp.dat中的温度数据读入MATLAB的变量中，在pdxThead.dat中也有相同的数据，只是它还包含第一行的列标题"month"、"high"、"low"和"ave"。下面的语句从文件pdxThead.dat中读取温度数据：

```
>> fid = fopen('pdxThead.dat','rt');  % Open pdxThead.dat for reading
>> headings = fgetl(fid)              % Read line containing column headings
headings =
month  high    low     ave

>> d = fscanf(fid,'%f');              % Read all data into the d vector
>> fclose(fid)                        % Close the file
>> d = reshape(d,[4 12])'             % Convert to a 12-by-4 matrix,
                                      % note transpose
data =
     1.0000   45.3600   33.8400   39.6000
     2.0000   50.8700   35.9800   43.4300
     3.0000   56.0500   38.5500   47.3000
     4.0000   60.4900   41.3600   50.9200
     5.0000   67.1700   46.9200   57.0500
     6.0000   73.8200   52.8000   63.3100
     7.0000   79.7200   56.4300   68.0700
     8.0000   80.1400   56.7900   68.4700
     9.0000   74.5400   51.8300   63.1800
    10.0000   64.0800   44.9500   54.5200
    11.0000   52.6600   39.5400   46.1000
    12.0000   45.5900   34.7500   40.1700

>> m = d(:,1);           %  copy month data into the m vector
>> T = d(:,2:4);         %  copy temperature data into the T matrix
```

使用下列语句也可以达到相同的效果，如：

```
>> fid = fopen('pdxThead.dat','r');
>> headings = fgetl(fid);
>> d = fscanf(fid,'%f',[4 12])';    %  Read 48 numeric values and store in
                                    %  a 12-by-4 matrix, note transpose

>> fclose(fid)
>> m = d(:,1);     T = d(:,2:4);
```

（参考NMM工具箱中util目录下的loadColData函数，可获得从带文本头的文件中读取数据的更通用的解决方法。）

62

2.5 在MATLAB中绘制图形

MATLAB能够根据存储在向量或矩阵中的数据画出图形，可以通过计算解析函数或读取文件得到这些数据。基本的画线函数用于在一个绘图区域创建一条曲线$y=f(x)$或多条曲线$y_1=f(x_1)$、$y_2=f(x_2)$等。$z=f(x, y)$形式的二维数据能够用轮廓线和表面图表示。三维对象（如椅子、桌子、齿轮和人等）可以用几个阴影表面图来表示。每种图形都可以做成动画的形式。

本节将介绍基本的线、轮廓线和三维图形的创建。课后练习中需要用到一些其他的绘图函数。想更深入学习复杂图形的绘制方法，请参看文献[74]*Using MATLAB Graphics*和文献[51]。

2.5.1 画线

画线用于表示因变量作为自变量的函数的变化关系图。假设向量x和函数$y=f(x)$，画y-x线，用户可以输入语句：

```
>> x = ...        %  create x and y data
>> y = ...
>> plot(x,y)
```

系统默认是用plot函数将数据点用实线连接。如果要用空心圆标注数据点，用户可以输入下列语句：

```
plot(x,y,'o')
```

图2-5是画线的两个例子，在这两个例子中，也可使用其他的线型和符号来连接和标注数据点。函数plot对某个数据集画图的一般形式是：

```
plot(xdata,ydata,symbol)
```

这里*symbol*由表2-6中的符号指定。也可以同时指定符号使用的颜色和类型，如：

```
plot(x,y,'yo')
```

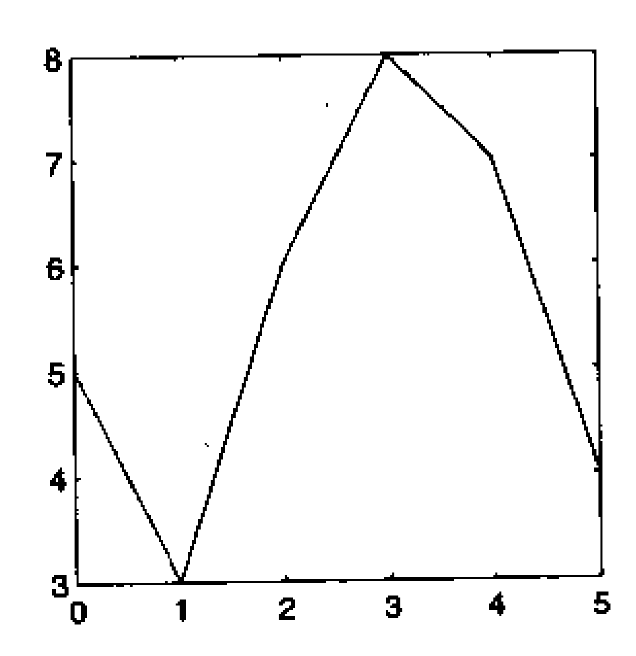
表示数据点是用黄色的空心圆标出。

63

```
plot(x,y,'r--')
```

```
x = [0 1 2 3 4 5];
y = [5 3 6 8 7 4];
plot(x,y)
```

```
x = [0 1 2 3 4 5];
y = [5 3 6 8 7 4];
plot(x,y,'o')
```



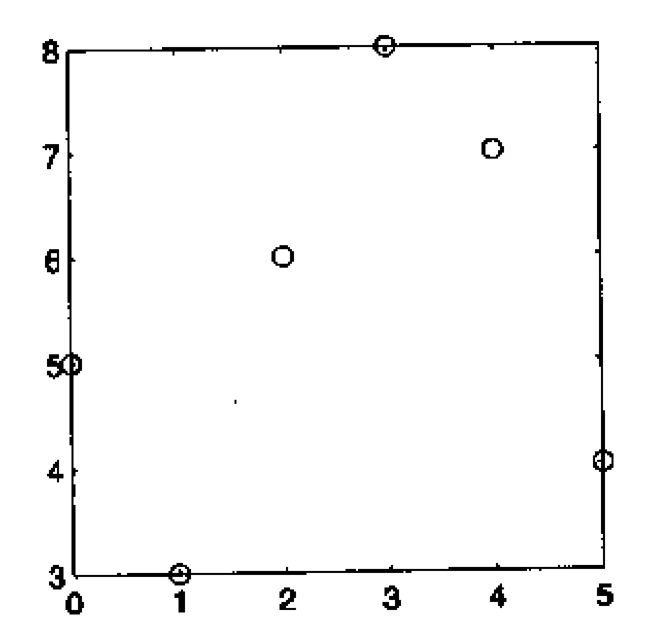


图2-5 将数据点用实线连接（左边）和只用符号标出数据点（右边）

表2-6 MATLAB 5中plot函数使用的指定的颜色、符号和线型

颜色		符号		线	
y	黄色	.	点	-	实线
m	洋红	o	空心圆	:	点线
c	青色	x	x标记	-.	点画线
r	红色	+	加号	--	虚线
g	绿色	*	星号		
b	蓝色	s	正方形		
w	白色	d	菱形		

（续）

颜色		符号		线
k	黑色	v	下三角	
		^	上三角	
		<	左三角	
		>	右三角	
		p	五角星形	
		h	六角星形	

[64]

表示用红色的虚线连接数据点。

```
plot(x,y,'k+--')
```

表示用黑色+号标出数据点并用黑色的虚线连接数据点。

函数plot可以在同一绘图区域中画出多个图形。例如，要显示3个向量对（x1,y1）、（x2,y2）和（x3,y3）构成的线条，可以输入下面语句：

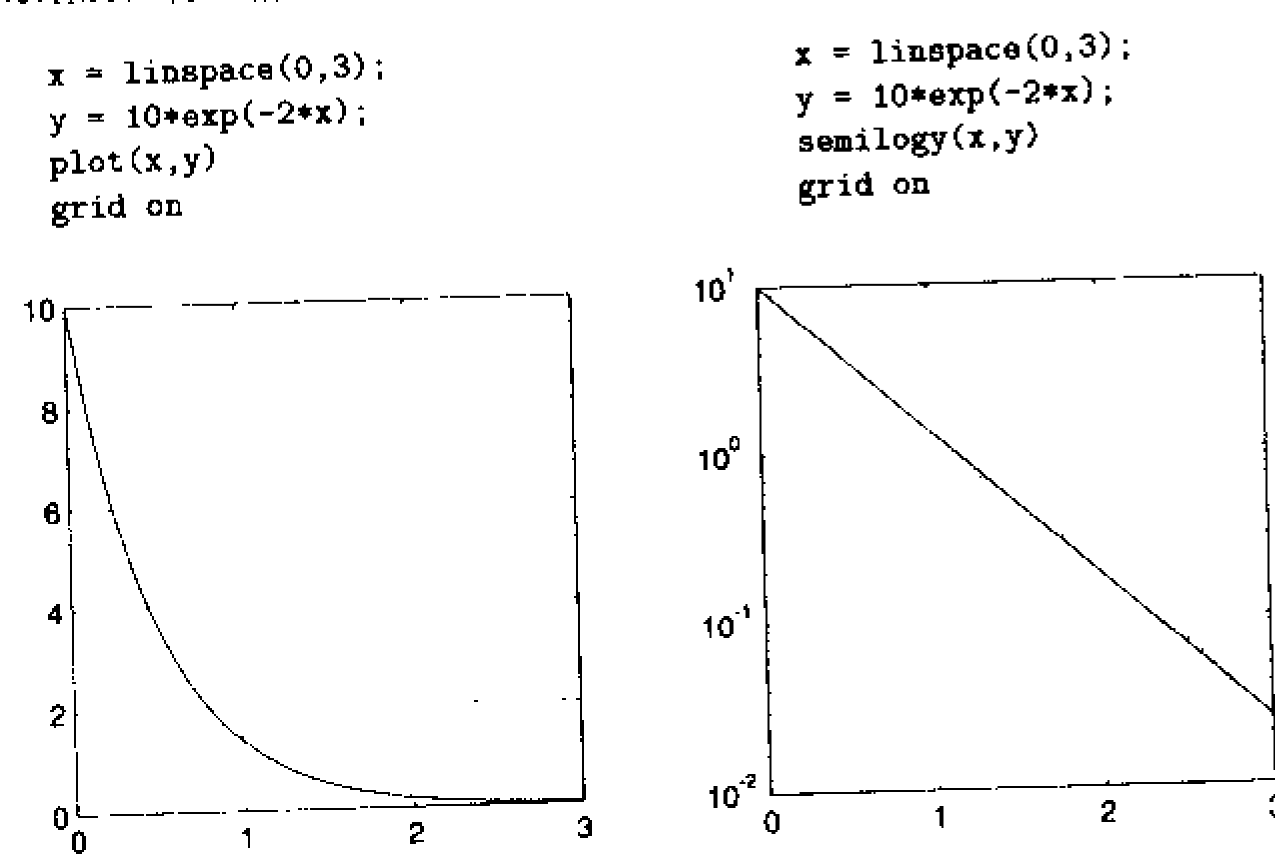
```
plot(x1,y1,x2,y2,x3,y3)
```

要改变各条曲线使用的符号，可以用字符串变量指定符号类型，如：

```
plot(x1,y1,s1,x2,y2,s2,x3,y3,s3)
```

这里s1、s2、s3是由表2-6中的符号构造的字符串。

对数和半对数图由函数loglog和semilogx、semilogy创建，且这些函数和函数plot所带参数相同。图2-6是使用semilogy画指数衰减图形的例子。为突出y轴的线性刻度和对数刻度的差别，将网格线用grid函数显示出来，这个函数将在下面介绍。



[65]

图2-6 指数函数分别在线性轴上的图形和在半对数轴上的图形

2.5.2 给图形作注解

坐标、图例和其他的注解能极大地提高图形的表达能力。表2-7中的函数可以用于添加这些注解。

表2-7　给图形作注解的函数

注解函数	执行的操作
axis	规定x轴和y轴的最大值和最小值
grid	依照x和y轴中的主刻度来画网格线
gtext	在对应鼠标输入处添加文本
legend	一个图形上有多条曲线时，用于标明不同的符号和线型的含义
text	在点（x,y）添加文本
title	在图形顶部添加标题
xlabel	用字符串标注x轴
ylabel	用字符串标注y轴

函数xlabel、ylabel和title分别给x轴、y轴和图形顶部添加图形的标注。这些函数的参数是文本字符串。系统会自动布局这些标注所处的位置。

函数legend将图例放在图形上以标明曲线中符号和线型的含义。它的语法如下所示：

```
legend(label1,label2,label3,...)
legend(label1,label2,label3,...,position)
```

这里"*label1,label2,...*"是前面调用函数plot（或等价函数）时用到的符号和线型。可选参数*position*表示图例在图形中的位置。MATLAB默认将图例放在恰当的位置以便不影响图形的显示。

例如，根据一个物体的位置x，速度v和加速度a画出它们随时间变化的关系图。下面的语句用于创建它们的图形区和对应数据的图例：

66

```
>> t = ...      % define time data
>> x = ...      % position
>> v = ...      % velocity
>> a = ...      % acceleration
>> plot(t,x,'-',t,v,'*',t,a,'o')
>> xlabel('time  (seconds)')
>> legend('position','velocity','acceleration')
```

函数legend中标签参数的顺序必须和函数plot定义的数据集顺序相同。

例2.6　注解函数的使用

在例2.4中，将俄勒冈州波特兰市机场的温度数据从文本文件中读取并对这些数据进行分析。下面的语句对这些数据画图（用函数xlabel、ylabel、title、axis和legend对图2-7中的图例注解）：

```
load pdxTemp.dat;       %  Read data into pdxTemp matrix
m = pdxTemp(:,1);       %  Copy first column into m (month) vector
t = pdxTemp(:,2:4);     %  and remaining columns into t (temperature) matrix

%  Create the plot with different symbols for each data set
plot(m,t(:,1),'ro',m,t(:,2),'k+',m,t(:,3),'b-');

xlabel('Month');                        %  Add axis labels and plot title
ylabel('Temperature ({}^\circ F)');
title('Monthly average temperature for Portland International Airport');
axis([1 12 20 100]);
legend('High','Low','Average',2);       %  Create the legend
```

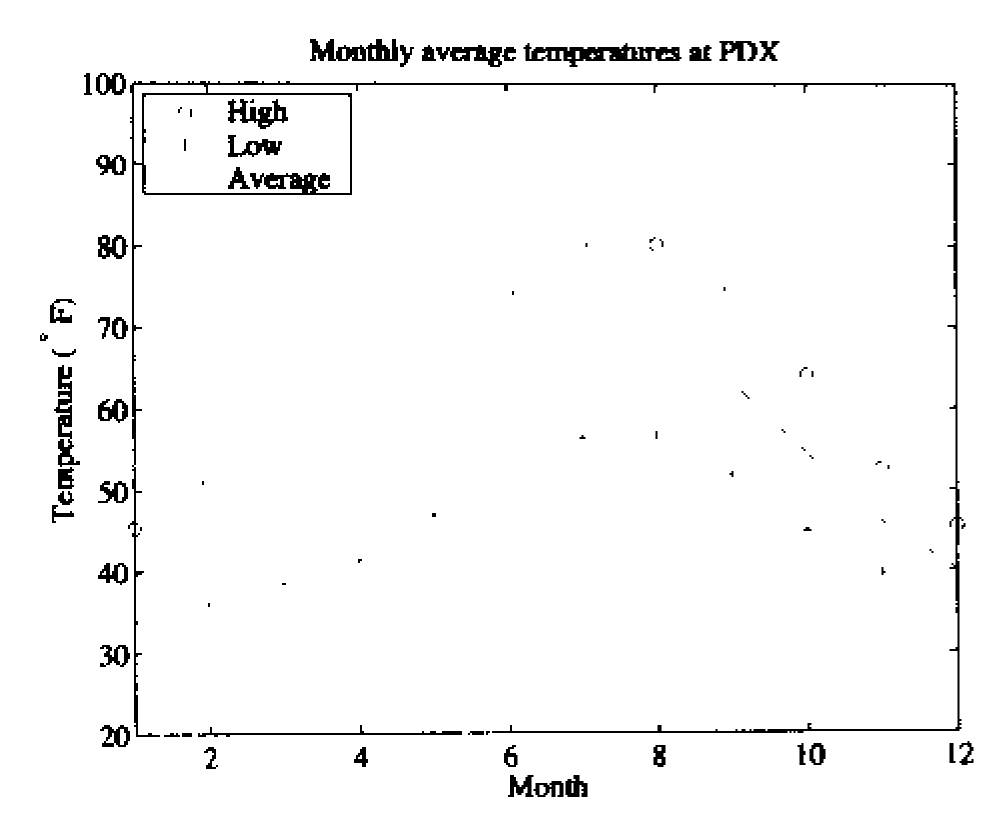


图2-7 使用注解函数对俄勒冈州波特兰市机场每月的平均气温变化图作注解

67

y轴的温度单位采用TEX符号{}^\circ。请读者参照文献[74]*Using* MATLAB *Graphics*关于在注释时使用$\mathrm{T_EX}$符号的详细信息。

2.5.3 子视窗

通常情况下，需要在一个图形视窗中画出多个图形。使用函数subplot能够达到这一目的，函数subplot带3个参数，如下所示：

```
subplot(nrows,ncols,thisPlot)
```

这里*nrows*和*ncols*表示图形视窗中有*nrows* × *ncols*个子视窗，*thisPlot*表示现在将要在其中绘制图形的子视窗的编号，视窗的编号是一个整数，它在图形中先按行、再按列排列。给定图形视窗中子视窗的布局，*nrows*和*ncols*的值不变。函数*subplot*在图形的布局确定之前给参数*thisPlot*赋适当的值。图2-8是用下面的语句画出的4个子图：

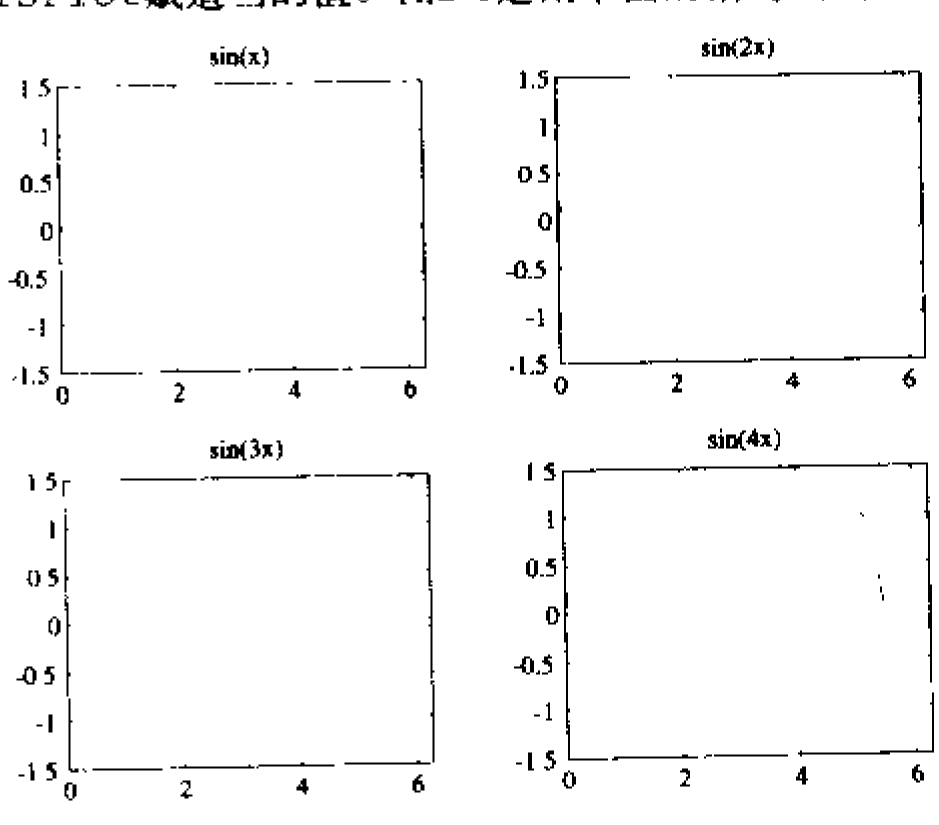


图2-8 一个图形视窗中的4个子视窗

68

```
>> x = linspace(0,2*pi);
>> subplot(2,2,1);
>> plot(x,sin(x));      axis([0 2*pi -1.5 1.5]);   title('sin(x)');

>> subplot(2,2,2);
>> plot(x,sin(2*x));    axis([0 2*pi -1.5 1.5]);   title('sin(2x)');

>> subplot(2,2,3);
>> plot(x,sin(3*x));    axis([0 2*pi -1.5 1.5]);   title('sin(3x)');

>> subplot(2,2,4);
>> plot(x,sin(4*x));    axis([0 2*pi -1.5 1.5]);   title('sin(4x)');
```

2.5.4 绘制表面图

带两个独立自变量的标量函数$z = f(x, y)$定义了三维空间中的一个表面，用图形描绘这个表面有很多种方法。函数mesh、meshc、surf、surfc和surfl分别用于创建不同外观的表面图。函数mesh和meshc绘制网状图，函数surf、surfc和surfl可使表面表示成为根据不同亮度和阴影着色的小平面的集合。函数meshc和surfc结合使用可以绘制在所画表面图的下部附加z=常数的轮廓图。在函数mesh和surf最基本的使用形式中，它们带相同的输入参数。这里只详细介绍函数surf。

在使用画表面图的函数之前，有必要试试如下两个内置绘图函数：

```
>> graf2d2
>> graf3d
```

众所周知，图形比文字更有实用价值，三维图形和MathWorks绘出的图形也都有这一特点。

函数surf有下面几种调用方式：

```
surf(Zdata)
surf(xvec,yvec,Zdata)
surf(Xmat,Ymat,Zdata)
```

函数$z=f(x, y)$的值存储在*Zdata*矩阵中，它由网格中的向量*xvec*和*yvec*或矩阵*Xmat*和*Ymat*定义。若网格由向量*xvec*和*yvec*定义，表面函数$z=f(x, y)$对应于

69

```
Zdata(i,j) = f (xvec(i),yvec(j))
```

这要求网格是矩形的。若一个网格由矩阵*Xmat*和*Ymat*定义，表面函数$z=f(x, y)$对应于

```
Zdata(i,j) = f (Xmat(i,j),Ymat(i,j))
```

当网格没有指定时（如，使用surf(*Zdata*)），默认定义网格为xgrid=1:m，ygrid=1:n，这里[m,n]=size(Zdata)。

函数surf和mesh可对外部数据或MATLAB中由解析函数计算所得数据进行绘图。在计算解析函数时，使用函数meshgrid能够方便地产生(x, y)数据对用来计算$z=f(x, y)$，语句如下所示：

```
[X,Y]= meshgrid(xg,yg)
```

创建了$m \times n$的矩阵X和Y

$$X = \begin{bmatrix} xg_1 & xg_2 & \cdots & xg_n \\ xg_1 & xg_2 & \cdots & xg_n \\ \vdots & \vdots & & \vdots \\ xg_1 & xg_2 & \cdots & xg_n \end{bmatrix} \qquad Y = \begin{bmatrix} yg_1 & yg_1 & \cdots & yg_1 \\ yg_2 & yg_2 & \cdots & yg_2 \\ \vdots & \vdots & & \vdots \\ yg_m & yg_m & \cdots & yg_m \end{bmatrix}$$

这里向量xg和yg分别定义了x轴和y轴的网格线。使用函数meshgrid和surf产生表面图如例2.7和例2.8所示。

例2.7 对解析函数绘制表面图

思考下面的二次方程

$$z=2-x^2-y^2 \tag{2-2}$$

其中，定义自变量的区间为$-5\leqslant x\leqslant 5$，$-5\leqslant y\leqslant 5$。要求画出$z=f(x, y)$的图形并满足：（1）定义(x, y)的网格；（2）在网格节点上计算z值；（3）使用恰当的surf或mesh函数画图，下面的语句用于画出图2-9中的表面图：

```
>> xg = linspace(-5,5,20);      % x grid vector (used for y also)
>> [X,Y] = meshgrid(xg,xg);     % Grid matrices on a square
>> Z = 2 - X.^2 - Y.^2;         % Vectorized evaluation of Z = f(X,Y)
>> surf(X,Y,Z)                  % Create the surface plot
>> xlabel('x');  ylabel('y');   % Add axis labels
```

70

后面的习题32将介绍另一种表面图。

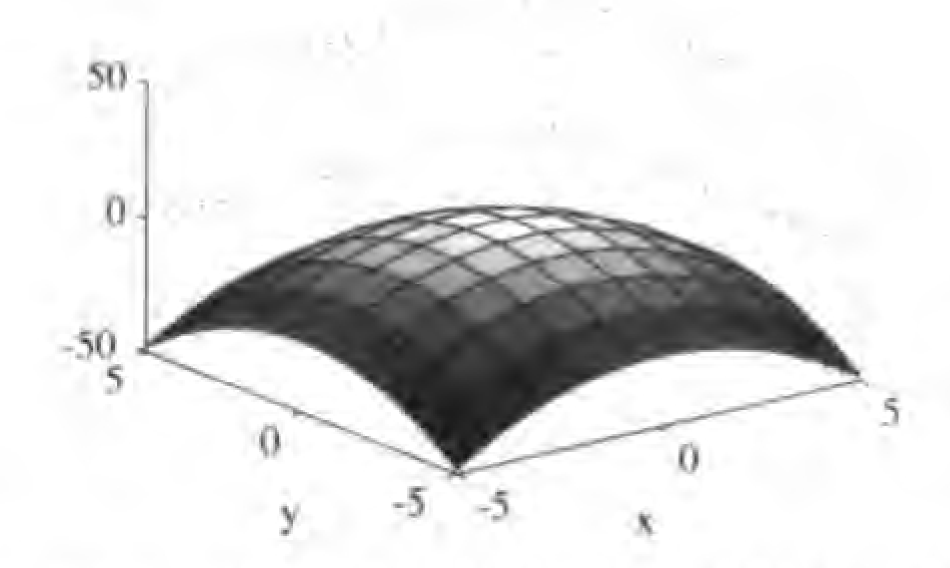


图2-9 $z=2-x^2-y^2$的表面图，它的定义域为$-5\leqslant x\leqslant 5$，$-5\leqslant y\leqslant 5$

例2.8 绘制其他类型的表面图

用下面的语句可以绘制函数$z=2-x^2-y^2$对应的四种类型的表面图：

```
>> x = linspace(-5,5,20);         % x grid vector (used for y also)
>> [X,Y] = meshgrid(x,x);         % Grid matrices on a square
>> Z = 2 - (X.^2 + Y.^2);         % Vectorized evaluation of Z = f(X,Y)

>> subplot(2,2,1);  mesh(x,x,Z);     title('mesh plot');
>> subplot(2,2,2);  surf(x,x,Z);     title('surf plot');
>> subplot(2,2,3);  surfc(x,x,Z);    title('surfc plot');
>> subplot(2,2,4);  surfl(x,x,Z);    title('surfl plot');
```

图2-10列出了它们对应的表面图。内置函数graf2d2和graf3d提供了更多的示例。

视角和颜色映像(color map) 表面图的视角可以用函数view来改变

```
view(azimuth,elevation)
```

这里*azimuth*和*elevation*是观测者视线与(x, y, z)平面的夹角。图2-11显示了这两个夹角。方位角α是观测者与x轴在平面(x, y)的夹角。仰角γ是视线向量与z轴的夹角。画完表面图后，可以用函数view改变观测者的观察点而不改变定义表面图的数据。

默认的观察视角等价于

71

```
>> view(-37.5,15)
```

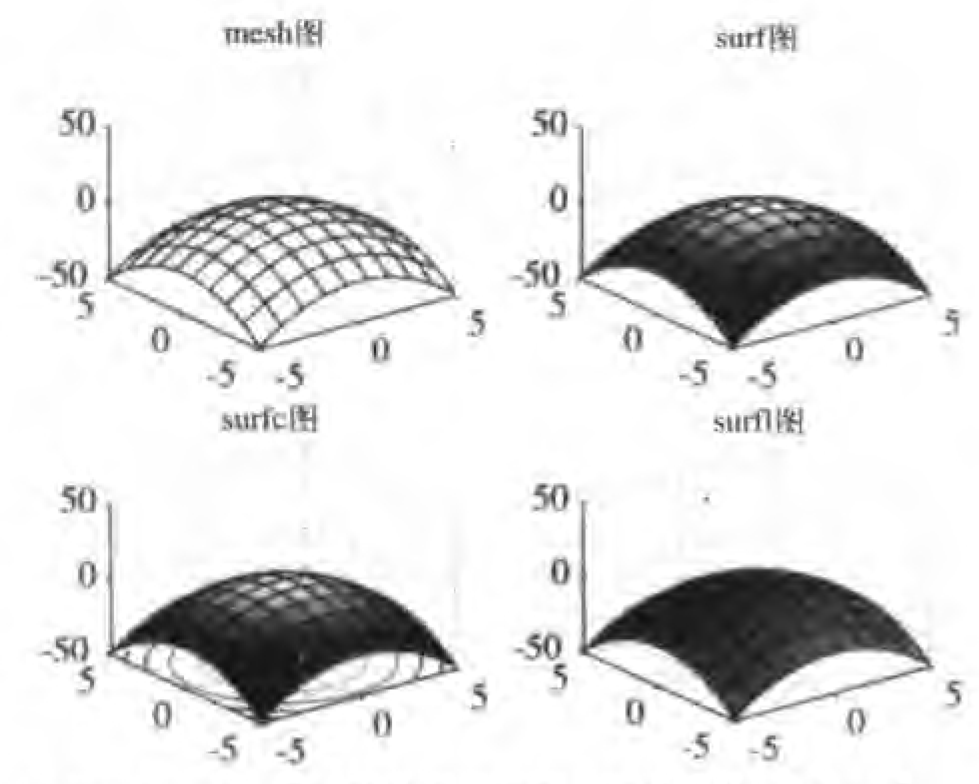
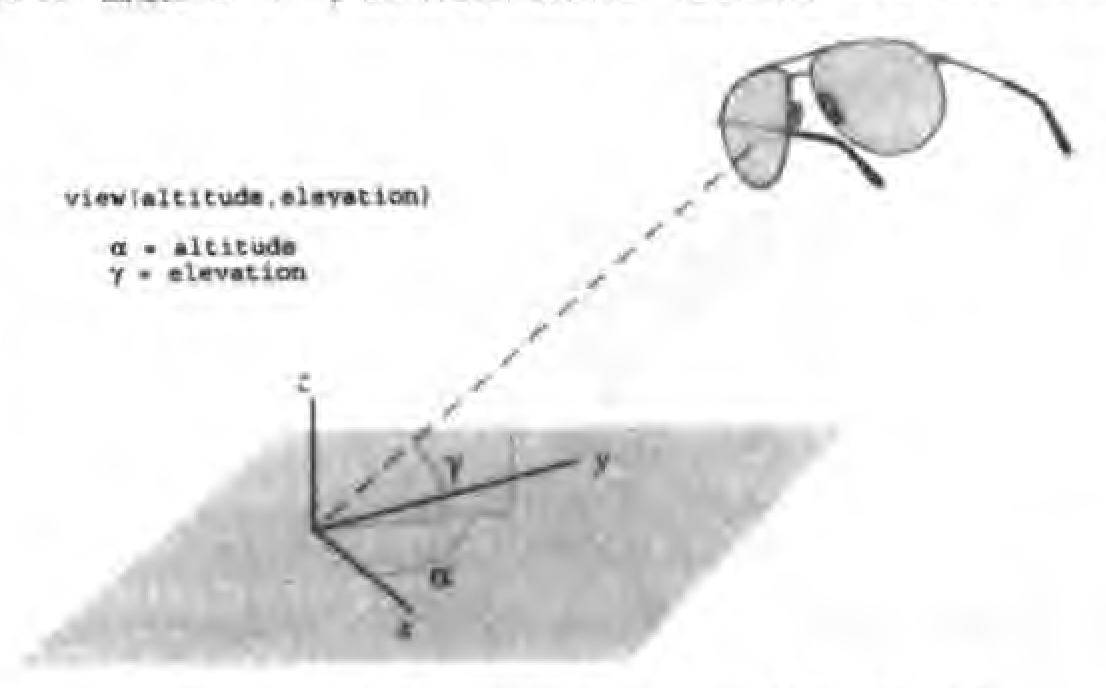


图2-10　函数$z=2-x^2-y^2$的4种类型表面图，定义域为$-5\leqslant x\leqslant 5$，$-5\leqslant y\leqslant 5$



72

图2-11　由方位角α和仰角γ定义的三维图形的视角

在创建完例2.7中的表面图后，试试下面的语句：

```
>> view(-10,15)
>> view(30,15)
>> view(30,0)
>> view(30,45)
>> view(30,60)
>> view(30,90)
```

除了改变视角外，用户可以通过改变网格分辨率、改变图形颜色和阴影从而改变表面图外观。对于方程（2-2）中的解析函数，要增加x和y的网格线的数量会增加画图的时间。增加分辨率将使表面图更加光滑，因为用来构成表面的各个小平面变小了。

*颜色映像(color map)*是一张表，它用于给函数`surf`的参数矩阵z中的每个元素添加颜色值。MATLAB中预先定义了颜色映像的字符串值，如'bone'、'cool'、'gray'、'hot'、和'jet'。例如：

```
>> colormap('hot')
```

表面图颜色会从暗红色到亮红色、再到黄色、再到白色（参照文献[74]*Using* MATLAB *Graphics*，或用"help graph3D"命令获得详细信息）。和使用view函数来改变视角一样，使用颜色映像不会改变定义表面图的数据。

使用下面的语句可以给表面图加阴影：

```
>> shading('flat')
>> shading('faceted')
>> shading('interp')
```

默认的阴影是"flat"，它给组成表面$z=f(x,y)$的小平面赋以统一的颜色值。"faceted"是在"flat"着色的基础上，给表面$z=f(x,y)$的每个小平面周围加上边线，以突出这些小平面。"interp"给每个小平面赋连续的颜色值。这样不需要增加小平面的个数，但是，能使所求平面更加光滑。

73

graf3d程序也支持表面图的渲染、着色和加阴影等操作。

2.5.5 轮廓线

轮廓线是函数$z=f(x,y)$的"水平横截面线"（即z=常量时的曲线）。传统的轮廓线是在(x,y)平面上的二维图形。然而，表面图直观地显示了$z=f(x,y)$的数据，轮廓线能够提供更多定量的信息。地形图是轮廓线最普通的应用，它为陆地导航提供了大量精确的信息。地形图上的轮廓线代表的是海拔高度常量。看地形图有经验的人能够根据地形图中的轮廓线看出某条轮廓线所代表的海拔高度。

函数contour有下面几种形式：

```
contour(Zdata)
contour(Zdata,ncont)
contour(Zdata,cvals)
contour(Xgrid,Ygrid,Zdata)
contour(Xgrid,Ygrid,Zdata,ncont)
contour(Xgrid,Ygrid,Zdata,cvals)
```

函数$z=f(x,y)$存储在矩阵Zdata中，它是由Xgrid和Ygrid构成的网格定义的。和函数surf一样，向量和矩阵能够定义网格。ncont定义轮廓线的条数。或者，用户可以指定向量cvals的值，它包含了要在该处绘制轮廓线的z的精确值（在线帮助和参考文献[75]MATLAB *Language Reference Manual*对轮廓线的变量有详细的介绍）。

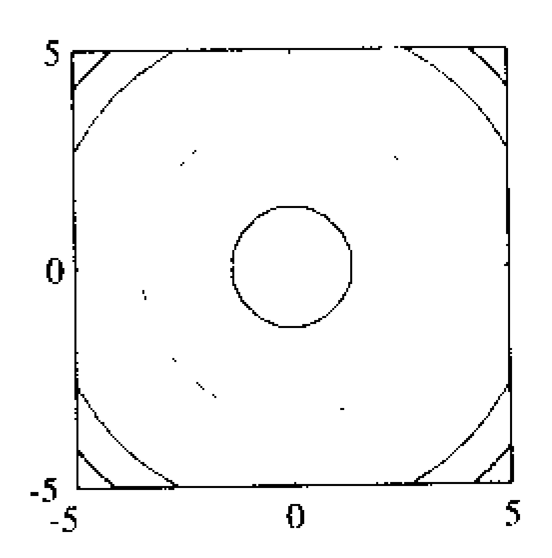


图2-12 函数$z=2-x^2-y^2$的轮廓线，定义域为 $-5<x<5$，$-5<y<5$

例2.9 解析函数的轮廓线

方程（2-2）定义的轮廓线与图2-9画出的表面图包含相同的信息。下列语句用于画出图2-12的轮廓线图。

74

```
>> xg = linspace(-5,5,20);      % Grid vector
>> [X,Y] = meshgrid(xg,xg);     % Create matrices of X and Y values
>> Z = 2 - X.^2 - Y.^2;         % Vectorized evaluation of Z = f(X,Y)
>> contour(X,Y,Z)               % Create the contour plot
>> axis('square')               % True aspect ratio for axis
```

轮廓线是圆，因为z=常量，即x^2+y^2=常量。课后练习34介绍了画轮廓线的其他方法。

例2.10 源-漏对（source-sink pair）的可视化

拉普拉斯(Laplace)方程

$$\frac{\partial^2\varphi}{\partial x^2}+\frac{\partial^2\varphi}{\partial y^2}=0$$

在静电学、电磁学和理想流体学有极大的应用潜力。拉普拉斯方程的基础解系可用于构造通解。基础解系之一就是图2-13所示的源-漏对。源-漏对的电位是：

$$\varphi=\frac{\Gamma}{2\pi}[\ln R_{si}-\ln R_{so}]$$

这里Γ是源和漏的强度，R_{si}是漏与点(x, y)之间的距离，R_{so}是源与点(x, y)之间的距离。

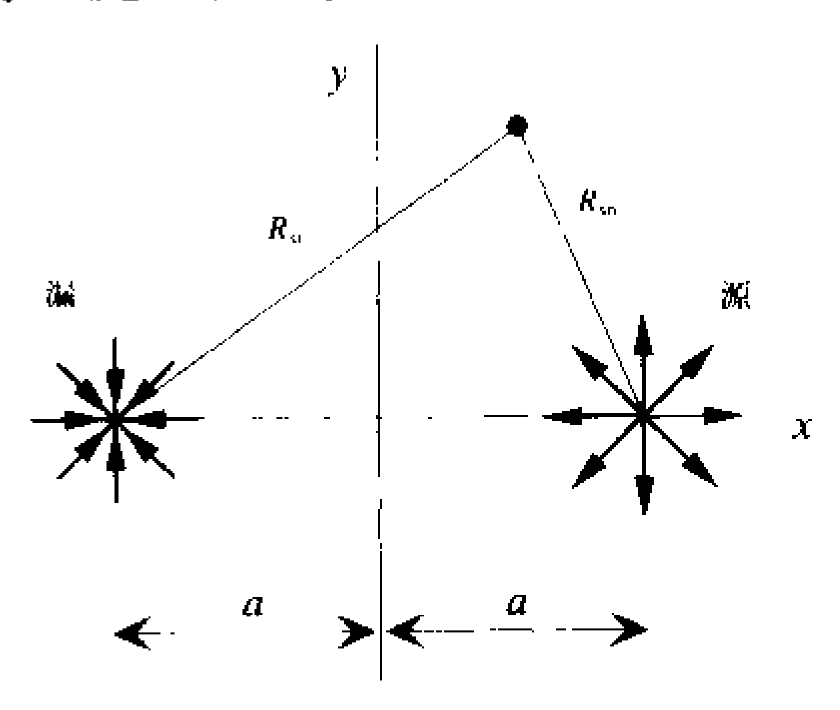


图2-13 (x, y)坐标平面上的源-漏对

显然：

$$R_{si}=\sqrt{(x+a)^2+y^2} \quad 且 \quad R_{so}=\sqrt{(x-a)^2+y^2}$$

75

漏和源与原点的距离都是a，它们关于原点对称。下面的语句分别绘制了函数$\varphi(x, y)$的表面图和轮廓线，如图2-14所示：

```
>> ngrid = 40;                          % Number of grid lines
>> xg = linspace(-5,5,ngrid);           % Vector of both x and y grid locations
>> [X,Y] = meshgrid(xg,xg);             % Prepare for vectorized R calculations
>>
>> gam = 10;                            % Strength of the source and sink
>> a = 1.5;                             % Distance from origin to source-sink
>> Rsi = sqrt( (X+a).^2 + Y.^2 );                % Distance from sink
>> Rso = sqrt( (X-a).^2 + Y.^2 );                % Distance from source
>> phi = (gam/(2*pi))*(log(Rsi) - log(Rso));   % Potential function

>> subplot(2,1,1)                       % First subplot for surface plot
>> surf(X,Y,phi)                        % Create surface plot
>> view(-20,15)                         % Adjust the viewing angle

>> subplot(2,1,2)                       % Second surface plot for contours
>> levels = -3:3;                       % Contour levels to be drawn
```

```
>> cs = contour(xg,xg,phi,levels);  % Create contours
>> clabel(cs,levels);               % and label them
```

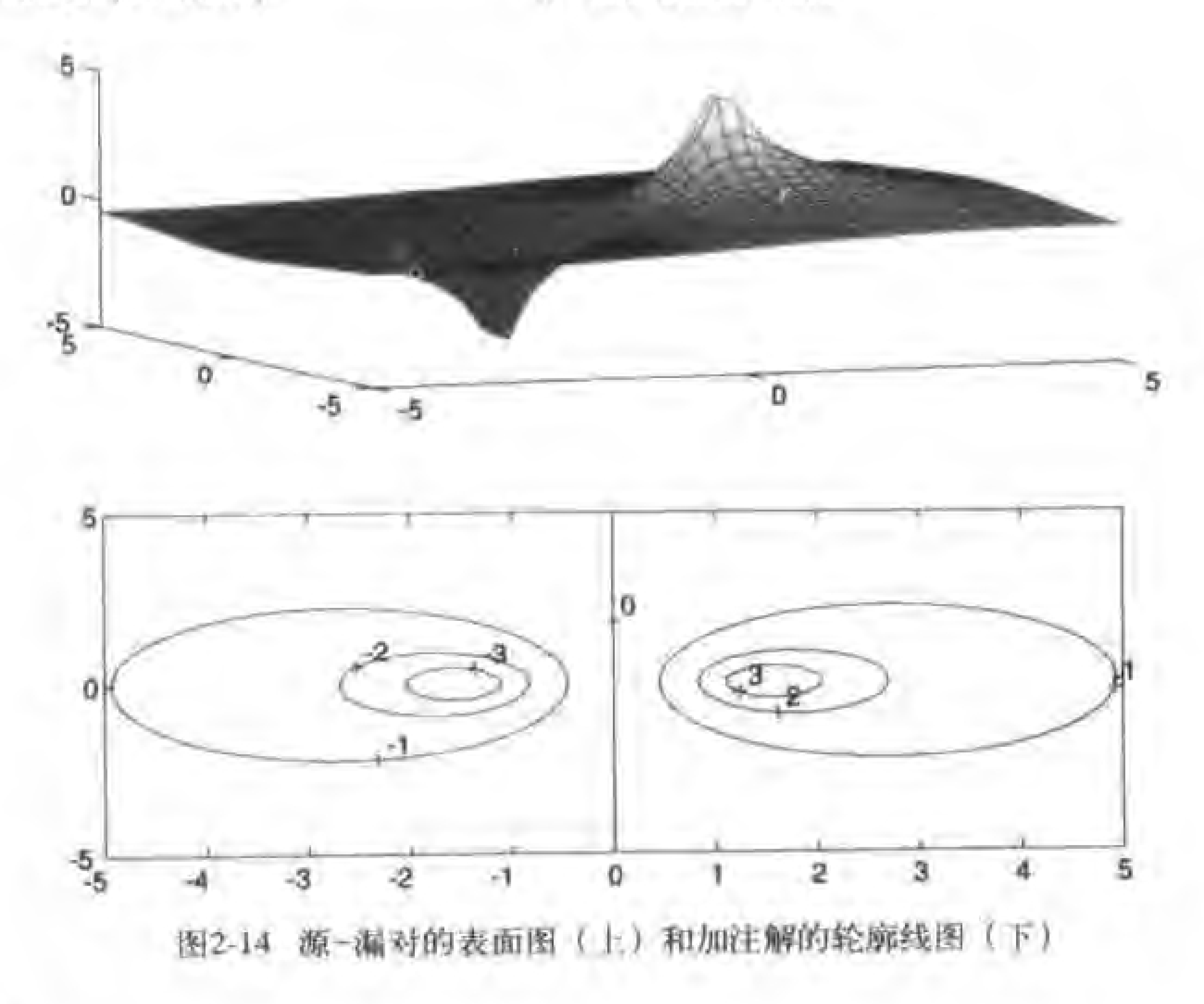


图2-14 源-漏对的表面图（上）和加注解的轮廓线图（下）

76

2.6 小结

本章介绍了MATLAB进行交互计算的主要操作。读者最好边做实验边学习使用MATLAB。想要更进一步深入学习MATLAB函数，请参照文献[73]*Using* MATLAB、[75]MATLAB *Language Reference*和参考图书[34]和[51]。

用户编制程序的时候可以调用本章介绍的所有的交互函数，这是下一章要介绍的内容。

习题

注意：大部分课后练习用到了本章未讲的内置函数，目的在于要求读者通过在线帮助和查找手册来学习新的内置函数的使用。

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量（参照1.4节中关于分级系统的介绍）。

1. *（1）使用`lookfor`函数查找与"max"相关的函数，再从返回的函数列表中使用`help`命令找到能获得矩阵的最大值的函数。使用所得函数查找下面两个矩阵的最大值：

$$A=\begin{bmatrix}1 & -5 & -2\\ 3 & 4 & -9\\ -7 & 2 & 6\end{bmatrix};\qquad B=\begin{bmatrix}\sin(1) & \sin(-5) & \sin(-2)\\ \sin(3) & \sin(4) & \sin(-9)\\ \sin(-7) & \sin(2) & \sin(6)\end{bmatrix}$$

2. （1）使用MATLAB内置函数计算下面的值：

(a) cosh(5)

(b) sinh(−2)

(c) $(e^5+e^{-5})/2$

(d) erf(1.2)，其中，$\mathrm{erf}(x)=\frac{2}{\sqrt{\pi}}\int_0^x e^{-\eta^2}\,d\eta$ 是误差函数

(e) $\beta(1,2)$，其中，$\beta(m,n)=\int_0^1 x^{m-1}(1-x)^{n-1}\,dx$

(f) $\beta(0.4, 0.7)$

(g) $J_0(2)$，其中，$J_0(x)$是第一种零阶Bessel函数

(h) $Y_0(2)$，其中，$Y_0(x)$是第二种零阶Bessel函数

3.（2）使用冒号运算符创建下面各表达式的对应向量，必要时可以用多条语句来实现。在不打印出矩阵的元素的情况下，使用内置函数norm判断两个矩阵是否相等（参看7.1.2节中向量范数的介绍）。

(a) `x=linspace(0,10,5)`

77

(b) `x=linspace(-5,5)`

(c) `x=logspace(1,3,3)`

(d) `x=logspace(1,3,5)`

4.（1+）同上题，要求创建列向量而不是行向量。

5.*（1+）使用linspace函数创建下面表达式的对应向量，必要时可以用多条语句（在不打印出矩阵元素的情况下，使用内置函数norm判断两个矩阵是否相等）。

(a) `x=0:10`

(b) `x=0:0.2:10`

(c) `x=-12:12`

(d) `x=10:-1:1`

6.（1+）同上题，要求创建列向量而不是行向量。

7.（1+）使用函数logspace编写一条语句创建向量x = [250,2500,25000,250000]。

8.（1+）使用MATLAB语句完成例2.1的运算。创建向量x、s、c和t构成三角函数矩阵[x s c t]并按表的形式显示所得三角函数值。能否只用一个转置符号达到要求？能否不用转置符号达到要求？

9.*（2 ）给定行向量x= [10 9 8 7]和列向量

$$y=\begin{bmatrix}1\\2\\3\\4\end{bmatrix}$$

至少使用两种不同的方法求行向量z，其中$z_i=x_i-y_i$。你的回答只需一条赋值运算，要求只能进行一次赋值操作，不必写出向量z中所有元素的方程。

10.（1）编写MATLAB语句来手工输入矩阵$\boldsymbol{A}$

$$\boldsymbol{A}=\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix}$$

然后从矩阵$\boldsymbol{A}$得到如下的矩阵$\boldsymbol{B}$

$$B=\begin{bmatrix}7&8&9\\4&5&6\\1&2&3\end{bmatrix}$$

要求不能手工输入矩阵B的值（提示：用“lookfor flip”）。

11. （1）矩阵可以看成是行向量和列向量的组合。给定行向量$u=(1,2,3)$和$v=(4,5,6)$。用（一条）语句求2×3矩阵A。其中，u是第一行，v是第二行。

12. （1）给定矩阵

$$C=\begin{bmatrix}11&5\\2&1\\18&7\end{bmatrix}$$

用两行语句从矩阵C中引用列向量得出$s=(11,2,18)^T$和$t=(5,1,7)^T$。

13. （1）使用diag函数和冒号运算符创建下面矩阵

$$\text{(a)}\begin{bmatrix}1&0&0&0\\0&2&0&0\\0&0&3&0\\0&0&0&4\end{bmatrix}\quad\text{(b)}\begin{bmatrix}4&0&0&0\\0&3&0&0\\0&0&2&0\\0&0&0&1\end{bmatrix}\quad\text{(c)}\begin{bmatrix}1&0&0&0\\0&7&0&0\\0&0&13&0\\0&0&0&19\end{bmatrix}$$

赋值语句中不能有0。

14. （1+）使用diag函数创建$n\times n$对称三对角矩阵

$$D=\begin{bmatrix}2&-1&0&0\\-1&2&-1&0\\0&-1&2&-1\\0&0&-1&2\end{bmatrix}$$

（提示：使用函数diag带两个参数的形式。）

15. *（1）使用函数eye和fliplr创建矩阵

$$E=\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix}$$

使用函数flipud可以吗？

16. （2+）用一条语句创建如下矩阵：

$$\begin{bmatrix}1&0&0&0&1\\-1&1&0&0&1\\-1&-1&1&0&1\\-1&-1&-1&1&1\\-1&-1&-1&-1&1\end{bmatrix}$$

（提示：一种方法是使用矩阵相加以及内置命令tril、ones、eye和zeros。）

17. （1）使用reshape函数和冒号运算符创建下列矩阵

$$(a)\ \begin{bmatrix}2 & 8 & 14 & 20\\ 4 & 10 & 16 & 22\\ 6 & 12 & 18 & 24\end{bmatrix}\quad (b)\ \begin{bmatrix}-5 & -3 & -1 & 1 & 3 & 5\\ -4 & -2 & 0 & 2 & 4 & 6\end{bmatrix}$$

可先用冒号运算符创建向量，再用reshape函数创建所要得到的矩阵，也可用一个表达式完成。

79

18. （2　）思考下面的程序：

```
>> A = ones(3,2);  B = 2*ones(2,3);  A*B;
>> A(2,3) = 2;
>> A*B
??? Error using ==> *
Inner matrix dimensions must agree.
```

为什么第一句A*B是合法的，而第二句不合法？它是由哪条语句造成的错误？假如是输入错误，请更正。

19. （2）解释下列程序中使用.*数组运算符造成的错误：

```
>> u = 0:3;  v = (3:-1:0)';
>> w = u.*v
??? Error using ==> .*
Matrix dimensions must agree.
```

为什么维数不等将导致矩阵不能点乘？如果不采用循环创建列向量w，怎样可以使得w(i) = u(i)*v(i)。

20. （2）对下列每个矩阵

$$\boldsymbol{C}=\begin{bmatrix}(9+2) & (-2-2)\\ (3-1) & (1+1)\\ (-3+4) & (7+4)\end{bmatrix}\quad \boldsymbol{D}=\begin{bmatrix}(9-2^2) & (-2-2^2)\\ (3-1) & (1-1)\\ (-3-4^2) & (7-4^2)\end{bmatrix}$$

$$\boldsymbol{E}=\begin{bmatrix}\frac{9}{2} & \frac{-2}{-2}\\ \frac{3}{-1} & \frac{1}{1}\\ \frac{-3}{4} & \frac{7}{4}\end{bmatrix}$$

（a）找出每个矩阵各列中绝对值最大的元素。

（b）找出每个矩阵中绝对值最大的元素。

（c）找出每个矩阵中绝对值最小的元素。

注意：矩阵$\boldsymbol{C}$、$\boldsymbol{D}$和$\boldsymbol{E}$由下列矩阵运算得到：

$$\boldsymbol{A}=\begin{bmatrix}9 & -2\\ 3 & 1\\ -3 & 7\end{bmatrix}\quad \boldsymbol{B}=\begin{bmatrix}2 & -2\\ -1 & 1\\ 4 & 4\end{bmatrix}$$

21. （1+）用MATLAB程序创建由$y=\mathrm{erf}(\alpha x)$、$0\leqslant x\leqslant 5$和α=0.1,0.3,0.5,0.7,0.9,1.1构成的表。用联机帮助可以获得关于erf函数更多的信息，并排列所求表使行对应x的值，列对应α的值。为了使所求表简短，选择10x值。

80

22. (2) 为什么下列程序中`A(k,k)`和`A(i,i)`表达式的出错信息各不相同?

```
>> clear all
>> A = ones(3,3);
>> A(k,k)
??? Undefined function or variable 'k'.

>> A(i,i)
??? Index exceeds matrix dimensions.
```

23. (1) 手工计算并验证第29页例子中关于变量`s`和`t`的复数运算的正确性。

24. * (1) 画出$\sin\theta$在$0 \leqslant \theta \leqslant 2\pi$区间上由60个数据点构成的图形。用空心圆标注点并用虚线连接各点(提示:为了把标注数据点的空心圆和曲线连起来,MATLAB 4的用户需要描两次数据点)。

25. * (1+) 分别画出$\zeta=0$、$\zeta=0.3$、$\zeta=0.9$时的二阶系统响应图。在同一个绘图区用例2.2中的公式连接对应三个ζ值的响应曲线。标注轴线并用图例标识对应的曲线。

26. (2+) 编写MATLAB程序画出$\tan\theta$在$-\pi \leqslant \theta \leqslant \pi$上的图形。若数据用两个向量存储(即`theta`和`tanTheta`),当$\theta=\pm\pi/2$时会发生什么情况?把数据分别存储在3个数组中并对这3组数据在同一坐标轴中画图能够避免错误。

27. (1+) 编写MATLAB程序画出y = `erf` (αx)的图形,其中$0 \leqslant x \leqslant 5$,$\alpha$ = 0.1,0.3,0.5,0.7,0.9,1.1。使用联机帮助可以得到`erf`函数的更多信息。重新布置图形区使x轴为水平轴,不同的曲线对应不同的α值。为使曲线更光滑,选择100x值。

28. (1+) 下表分别列出不同的温度下空气的粘度和水的粘度的变化,画出它们的对应关系。使用函数`plot`,`semilogy`和`loglog`哪个最好?为什么?数据存储在NMM工具箱的`airvisc.dat`和`H2Ovisc.dat`文件中。

T °C	μ_{air} kg/(m · s)	T °C	μ_{H_2O} kg/(m · s)
0	1.720×10^{-5}	0	1.787×10^{-3}
20	1.817×10^{-5}	5	1.519×10^{-4}
40	1.911×10^{-5}	10	1.307×10^{-4}
60	2.002×10^{-5}	20	1.002×10^{-3}
80	2.091×10^{-5}	30	7.975×10^{-4}
100	2.177×10^{-5}	40	6.529×10^{-4}
127	2.294×10^{-5}	50	5.468×10^{-4}
177	2.493×10^{-5}	60	4.665×10^{-4}
227	2.701×10^{-5}	70	4.042×10^{-4}
		80	3.547×10^{-4}
		90	3.147×10^{-4}
		100	2.818×10^{-4}

81

29. (2) 短跑运动员的理论模型(参考W. G. Prithard所编的*Mathematical Models of Runnning*,SIAM Review, vol. 35,No. 3,pp. 359-379,1993年9月)如下:

$$x(t)=a\tau[t-\tau(1-e^{-t/\tau})]$$

这里x是时间t内跑的实际距离，a和τ分别是对应具体的运动员和赛跑距离。卡尔刘易斯和本约翰逊在1987年罗马世界杯100米跑中的测量数据$x(t)$如下表所示：

x(m)	0	10	20	30	40	50	60	70	80	90	100
刘易斯 time(s)	0	1.94	2.96	3.91	4.78	5.64	6.50	7.36	8.22	9.07	9.93
约翰逊 time(s)	0	1.84	2.86	3.80	4.67	5.53	6.38	7.23	8.10	8.96	9.83

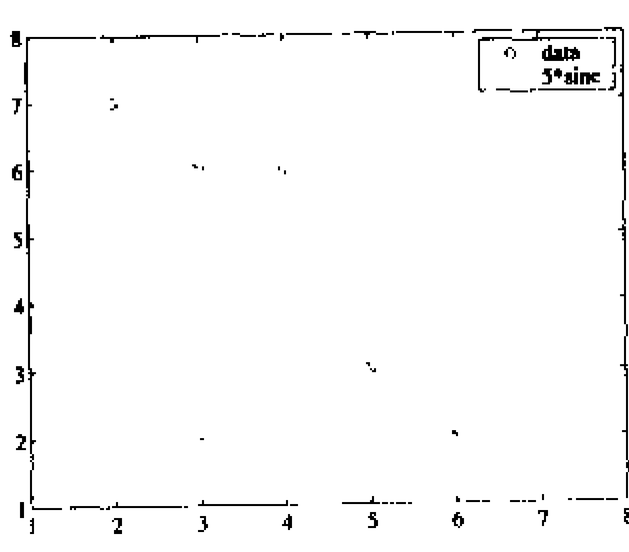
对刘易斯的测得数据进行曲线拟合得到τ= 0. 739s、a= 14. 4m/s^2. 在同一个图形区域中画出刘易斯和约翰逊的测量数据和刘易斯的理论模型图。理论图形用实线连接，测量数据用符号标注且不连成线。为使理论图形在小时间段光滑，可以用更短的时间间隔，如0.2s。标注坐标轴并在图形区中添加图例。数据存储在NMM工具箱的data目录下的sprint.dat文件中。

30. * (2) 下表是理论模型$y=5x\exp(-3x)$的试验数据。x_m和y_m由测量得来，δ_y的值由不确定性分析得来。使用内置errorbar函数给图形添加错误信息条。用hold on和plot函数覆盖由理论模型测得的数据所得的图形。数据保存在NMM工具箱的data目录下的xydy.dat文件中。

x_m	0.010	0.223	0.507	0.740	1.010	1.220	1.530	1.742	2.100
y_m	0.102	0.620	0.582	0.409	0.312	0.187	0.122	0.081	0.009
δ_y	0.0053	0.0490	0.0671	0.0080	0.0383	0.0067	0.0417	0.0687	0.0589

31. (2+) 对左边给定的MATLAB程序，需要添加什么语句才能得到右边的图形？不要添加多余的数据，不考虑坐标轴上和图例中的字体和字体大小。不要用MATLAB 5. 3或更高版本的交互画图工具。

```
x = [2  3  4  5  6  7];
y = [7  6  6  3  2  3];
z = 5*sin(x);
```



82

32. (1) 根据例2.7中的语句，用下列语句代替surf (X,Y,Z)画出下列语句对应的表面图：

(a) `mesh(X,Y,Z);`

(b) `meshc(X,Y,Z);`

(c) `surf (X,Y,Z); shading('interp');`

(d) `surfc(X,Y,Z); shading('flat');`

(e) `surfl(X,Y,Z);`

(f) `surfl(X,Y,Z); colormap('bone')`

33. (2+) 表面图不必定义在矩形网格上。画出由圆$x^2+y^2=5$所定义区域内的表面$z=2+x^2+y^2$。计算圆内的点对应的函数值，这由MATLAB的画图程序来完成。首先，用下列语句定义向量r和θ。

```
>> r = linspace(0,5,20);
```

```
>> theta = linspace(0,2*pi,10)
```

然后把它们外积并得到矩阵X和Y

$$X_{i,j} = r_i \cos(\Theta_j) \text{和} Y_{i,j} = r_i \sin(\Theta_j)$$

表面函数Z=f(x, y)的计算与例2.7一样。当X和Y为如下值时，比较新的图形与原来图形的异同:

```
>> x = linspace(-5,5,20);
>> [X,Y] = meshgrid(x,x);
```

34. (1) 根据例2.9中的函数，用下列函数代替contour (X,Y,Z) 画轮廓线:

(a)
```
minz = min(min(Z));  maxz = max(max(Z));
intervals = linspace(minz,maxz,25);
contour(X,Y,Z,intervals);
```
(b) `[c,h] = contour(X,Y,Z); clabel(c,h); colorbar('vert')`

(c) `contourf(X,Y,Z)`

(d) `contour3(X,Y,Z)`

(e) `meshc(X,Y,Z)`

(f) `surfc(X,Y,Z)`

83
~
84

第3章 MATLAB编程

本章内容包括MATLAB程序的结构和语法、程序中用到的新函数以及一些示例。这里假设读者已经掌握了基本的MATLAB函数和命令的使用方法。

MATLAB语言和其他高级语言很类似，它的语法类似于Fortran，并借鉴了C的一些语法。MATLAB程序有循环执行结构和条件执行结构，子程序类似于Fortran中的子程序和C语言的函数，子程序间通过输入参数和输出参数传递数据。非输入输出变量是子程序的内部变量。子程序间能够相互调用。

MATLAB的一些重要的特性使它有别于其他的高级语言：所有MATLAB程序都整合在一个交互环境中；MATLAB程序是解释执行，而不是编译执行；MATLAB程序中变量的数据结构都是矩阵；MATLAB动态管理内存，有利于算法的开发；它提供了最优化的矩阵加、减、乘运算的内置子程序，同时可以求解线性方程组和求特征值。

85

本章主题

1. m文件脚本

脚本文件能自动处理简单的作业。它没有输入和输出参数，并和命令工作区共享变量。本部分还讨论了脚本的创建和所受的限制

2. m文件函数

m文件函数是带输入输出参数的程序。这里介绍了m文件的语法

3 输入和输出

介绍用户输入和创建文本输出的函数

4. 流程控制

举例说明if条件语句结构、for和while循环结构及其相关概念

5. 向量化

描述用以代替for循环语句的向量化语句的使用方法，介绍向量化查找和替换操作的内置命令。

6. 解决方法

介绍解决复杂编程问题的特殊结构

图3-1 第3章的主题

图3-1总结了本章的组织结构。想要自己编程的读者需要对本章内容很熟悉。初读本章时可以略读最后两节"向量化"和"解决方法"，这两节内容在后面的章节中将会用到。只想学会如何运行程序的读者理解前两节"脚本"和"函数"就可以了。

3.1 m文件脚本

MATLAB程序包括脚本和函数两种。本章首先介绍脚本，然后介绍函数。脚本程序是保存在文件中的交互程序，在命令提示符后输入文件名就能运行。m文件脚本没有输入和输出参数，所以在固定任务中使用是很方便的。

86

例3.1 用脚本定义工程常量

工程学科中都存在物理常量集和单位间转换系数集，在设计计算时经常要用到常量集。若把这些常量定义在一个m文件脚本中，那么在MATLAB计算时，能够很快地查阅到这些常

量。程序清单3-1列出了脚本程序myCon作为示例。脚本myCon在NMM工具箱的program目录下，读者稍作修改就能把它应用到实际问题中去。

在命令提示符后输入文件名就能够运行脚本：

```
>> myCon
```

输入文件名后似乎没什么响应！事实上，脚本myCon中的语句执行后没有在命令窗口中输出任何信息，但脚本中定义的变量已添加到了工作区中（参照3.1.2节）。例如，执行完脚本myCon后，计算25公升等价于多少加仑就很便捷了：

```
>> 25*galPerLiter
ans =
    6.6043
```

程序清单3-1 用于保存工程分析中物理常量的MATLAB脚本文件

```
% myCon    Defines useful constants in the workspace
%          This must be a script, not a function m-file, in order to add
%          variables defined here to the workspace

% --- fundamental constants
Avagadro = 6.024e23;        % number of molecules in a mole
cLight = 2.998e8;           % Speed of light;  m/s
gravity = 9.807;            % acceleration of gravity,  m/s^2
hPlank = 6.625e-34;         % Plank's "h" constant;  J*s/molecule
kBoltzmann = 1.380e-23;     % Boltzmann's constant;  J/K/molecule
Rgas = 8315;                % Universal gas constant;  J/kmol/K

% --- unit conversions
galPerMeter3 = 264.17;                % number of US gallons in a cubic meter
galPerLiter = galPerMeter3/1000;      % number of US gallons in a liter
inchPerMeter = 1/25.4e-2;             % number of inches in a meter
inchPerGal = 231;                     % number of cubic inches in a gallon
lbPerkg = 2.2046;                     % number of pounds (mass) per kilogram
psiPerPascal = 14.696/101325;         % number of PSI per Pascal
```


文件myCon中的变量名可能比较难记，可以把脚本的内容显示在命令窗口内，这样便于实现变量调用：

```
>> type myCon          在命令窗口输出函数myCon
```

3.1.1 创建m文件

MATLAB脚本和函数必须存储在以“.m”为扩展名的纯文本文件中，这些文件就称为m文件。纯文本文件只识别由字母和数字构成的文字，不允许文字处理器在文件中自动创建特殊格式的字符。大部分的文字处理器能够把文件保存为纯文本格式文件（在“File”菜单中的“Save as”选项）。使用文字处理器来创建m文件有些大材小用，所以使用文本编辑器或程序编辑器会更方便些。在Unix或Linux系统中，emacs和vi编辑器可用于创建m文件。

Windows下的MATLAB第五版自带文本编辑器，它能够用彩色高亮度显示语言结构和注释文字。这为编辑和调试带来了极大的方便。选择“File”菜单下的“Open...”选项，用户就能用内置编辑器打开m文件。选择“File”菜单下的“New...”选项，用户就能创建新的m文件。根据计算平台的不同创建和编辑m文件的信息，请用户参照文献[73]*Using* MATLAB。

例3.2 查找外卖店电话号码的脚本程序

假如读者在深夜编程并需要吃点快餐，而恰好在不远处有几家外卖店，但是问题出现了，读者没有记住它们的订餐电话。程序清单3-2中的takeout文件解决了这个问题。takeout脚本定义了一串变量，用于存储各家外卖店的订餐电话。这时，读者只要在MATLAB命令提示符后输入takeout就能够得到订餐电话，而不用去查电话薄。用户需要把takeout.m文件创建到MATLAB路径下才能使用这个文件（参照2.4.2节）。

程序清单3-2 外卖店订餐电话的MATLAB脚本

```
% takeout  Script to display restaurant telephone numbers.
mandarin_cove = '221-8343'
pizza_express = '335-7616'
bernstein_deli = '239-4772'
big_burger = '629-4085'
burrito_barn = '881-8844'
tofu_palace = '310-7900'
```

88

选择MATLAB的“File”菜单下的“New”选项创建一个新的脚本，并把程序清单3-2中的内容输入编辑窗口[⊖]。首行以%开始，这是注释符，MATLAB不解释和运行注释语句。注释语句在3.1.3节介绍。输入电话号码时，注意要用单引号括起来（不是左引号（‘），而是右引号（’））。在工作目录下保存文件为takeout.m。

在MATLAB路径下保存文件takeout.m后，在命令提示符后输入takeout，打印结果如下（为了简明，这里删去了多余的空行）：

```
>> takeout

mandarin_cove =
221-8343
pizza_express =
335-7616
bernstein_deli =
239-4772
big_burger =
629-4085
burrito_barn =
881-8844
tofu_palace =
310-7900
```

课后练习还会讨论takeout脚本的变化。

例3.3 绘图函数的脚本

创建和注释MATLAB中的图形是很方便的，但有时也绘制不出读者想要的图形。本例显示脚本文件（或m文件函数）如何帮助绘制具有印刷质量的图形。首先，在脚本文件中保存产生简单图形的语句，然后逐渐精加工脚本文件，完善图形。每当改变脚本中的语句时，要重新运行脚本，并重新绘制图形。

⊖ 读者可以输入自己喜欢的外卖店的名称和电话。

脚本文件通过反复试验能够使计算更加精确。MATLAB语句保存在文件中，所以重新计算很方便。为了表达得更直观易懂，这里以常见的三角函数绘图为例。这种方法也适用于更复杂的解析函数和保存在文本文件中的实验数据。

假设要画sinθ、cosθ和sinθ cosθ的图形，其中$0 \leqslant \theta \leqslant 2\pi$。为了"先简单后精炼"，打开一个纯文本文件[⊖]并输入下列语句：

```
t = linspace(0,2*pi);
y1 = sin(t);
y2 = cos(t);
y3 = y1.*y2;      % notice the array operator, .*
plot(t,y1,'-',t,y2,'.',t,y3,'--');
```

把文件保存为"trigplot.m"，并在命令窗口输入：

89

```
>> trigplot
```

假设脚本trigplot输入没有错误，运行结果就如图3-2的上半部所示。我们发现，图中并没有标注坐标轴，并且cosθ曲线用点连起来后显示得不够清晰。为了纠正这个缺点，需要改变plot语句并添加注释。(保留变量t、y1、y2和y3的定义，替换plot语句，添加axis和legend语句。)

```
plot(t,y1,'-',t,y2,':',t,y3,'--');
axis([0 2*pi -1.5  1.5])
legend('sin(t)','cos(t)','sin(t)*cos(t)');
```

此时，在命令窗口输入下列语句：

```
>> close all; trigplot
```

语句close all将关闭所有的图形窗口，并在出现plot语句时强制MATLAB打开一个新的图形窗口。这保证了前面的脚本运行结果不会影响到现在的图形窗口。

最后，使用TEX符号用θ替换t，把坐标的字体改为Times，添加标题，字体也是Times，如：

```
legend('sin(\theta)','cos(\theta)','sin(\theta)*cos(\theta)')
xlabel('\theta (radians)','FontName','Times','FontSize',14)
title('Plot of simple trigonometric functions','FontName','Times',
      'FontSize',12)
```

然后再重新绘制图形

90

```
>> close all;        trigplot
```

运行结果在图3-2的下半部。程序清单3-3列出了trigplot脚本程序的最终版本。

程序清单3-3 通过编辑例3.3中脚本文件所产生数据的绘图脚本

```
% trigplot Script to plot sin(x), cos(x), and sin(x)*cos(x)

x = linspace(0,2*pi);     % generate data
y1 = sin(x);
y2 = cos(x);
y3 = y1.*y2;              % y3 = sin(x)*cos(x)
```

⊖ 在Windows系统下，选择File菜单下的"New→m-file"，Unix和Linux系统下，用文本编辑器打开一个纯文本文件。

```
plot(t,y1,'-',t,y2,':',t,y3,'--');
axis([0 2*pi -1.5  1.5])
legend('sin(\theta)','cos(\theta)','sin(\theta)*cos(\theta)')
xlabel('\theta (radians)','FontName','Times','FontSize',14)
title('Plot of simple trigonometric functions',...
       'FontName','Times','FontSize',12)
```

本例的重要思想在于m文件脚本可以逐渐优化结果。读者在编写和执行m文件时，不必透彻地了解其计算任务，当然，无计划地盲目编程也是徒劳的。第4章介绍了结构化MATLAB程序开发并逐步完善的简单策略。

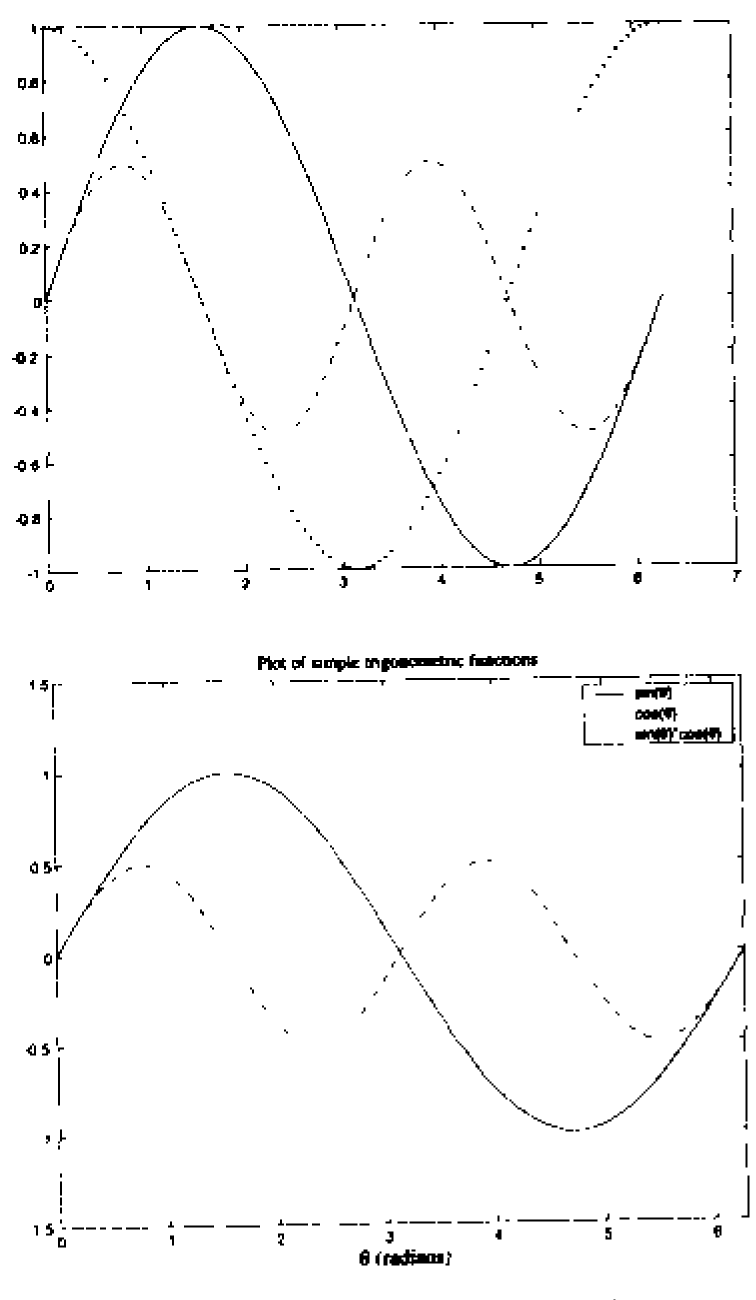


图3-2 改进脚本trigplot前后的图形

3.1.2 脚本的副作用

脚本中定义的所有变量都被添加到工作区中（参照2.4.1节）。这样，工作区中变量的值将被脚本中相同名字的变量值所改变，从而产生了运行脚本时的副作用（*side effect*）[⊖]。

91

仍然用例3.3的脚本trigplot作示例。当MATLAB路径下存在trigplot.m文件时，在

⊖ 除了把输入变量变成输出变量之外的任何动作称为副作用。事实上，当一个程序单元改变了不是它自己首先声明并最终销毁的某个变量的值后，就产生了副作用。一个普遍的例子是子程序改变全局变量的值。允许某些副作用的发生可使程序员编程更加灵活，但是，忽视副作用会造成程序错误，而且这种错误很难跟踪和排除。

命令窗口中输入下列语句:

```
>> clear
>> who                    (不显示变量名)
>> trigplot               (在工作区创建变量x、y1、y2和y3)
>> who

Your variables are:

x          y1          y2          y3
```

clear命令清除工作区中的所有变量。第一个who命令紧接clear命令，显示工作区中已经没有变量。执行完trigplot脚本后，画出图形并把变量x、y1、y2、y3添加到工作区。第二个who命令显示添加变量后工作区的结果。 92

假如trigplot是某个计算的一部分，如果存在已定义的变量x、y1、y2或y3，那么在运行完脚本trigplot后，将替换它们原来的值。如果要对更新后的x、y1、y2、y3进行运算，那么这种替换就是我们所希望的。例3.1中mycon脚本把变量添加到工作区中，就是利用了这一点。但是，在运行trigplot后并没有提示x、y1、y2、y3被替换。在后面的分析中由于变量x、y1、y2、y3的值被替换而可能会出现错误。

由于脚本文件会产生副作用，建议在数值分析时只使用m文件函数。创建m文件函数和脚本一样简单，而且更加安全，更加灵活。

3.1.3 注释语句

前面例子中的脚本文件已使用了注释语句来描述对应的代码，注释语句就是从%号开始到本行末尾的内容。短的注释语句可以和执行语句在同一行，多行注释语句在每行的开头都要加%号。

注释符号能够打开和关闭一段程序，这在调试或保存不用的可选步骤和过程以备参考的时候是很有用的。要使一段程序不运行，只要在程序开始处加上%号即可。

3.2 m文件函数

m文件函数是MATLAB中的子程序，它类似于Fortran中的子程序和C中的函数。用户自定义的MATLAB函数就像内置函数一样。MATLAB的大部分函数都是m文件函数[⊖]。函数是通过预定义的输入输出参数列表与命令窗口及其他函数进行通信的代码模块。像3.6.2节介绍的一样，使用全局变量可以在模块间交换数据。函数中定义的变量是局部变量，局部变量只能用在定义它的函数中。

函数具有模块化、结构化和重用性的特点。输入和输出参数使程序更加灵活。在第4章介绍的大项目中，代码模块化有利于工程中的调试和维护，所以，在许多数值计算任务中，使用m文件函数而不用脚本。 93

3.2.1 函数语法

m文件函数的第一行的格式是:

⊖ 参考MATLAB目录中的toolbox目录下的MATLAB程序。

```
function [outputParameterList] = functionName(inputParameterList)
```

这里的*inputParameterList*和*outputParameterList*中各个参数用逗号隔开，它为函数的输入和输出传递数据。和其他的编程语言不一样的是，MATLAB函数带两个参数表：一个是输入参数表，另一个是输出参数表。这更体现了MATLAB是对数据集操作的思想，即：对输入数据集进行操作然后返回输出数据集。

函数定义的第一行的第一个字必须是“function”，紧接着是用方括号[]括起来的输出参数。如果函数没有输出参数列表，可以省略等号及方括号；如果只有一个输出参数，方括号可要可不要。在3.2.2节的示例中列出了函数定义的各种形式。

“founctionName”是函数的名字，由字符串构成，可以用它来调用函数。它必须与已存在的文件同名(不带“.m”)。如，函数“foo”必须保存在“foo.m”文件中。文件名后的*inputParameterList*用圆括号括起来，输入参数列表可选。输入参数和输出参数由有效的MATLAB变量名组成。变量可以是任何的MATLAB数据类型，如向量、矩阵、字符串、结构体和单元阵列等。

3.2.2　输入和输出参数

[94] 程序清单3-4包含了带输入和输出参数的一系列函数。这些函数的计算没有特殊意义。它们分别保存在三个独立的m文件中：twosum.m、threesum.m和addmult.m。

程序清单3-4　带输入和输出参数的函数示例

twosum.m

```
function twosum(x,y)
% twosum  Add two matrices and print the result
x+y
```

threesum.m

```
function s = threesum(x,y,z)
% threesum  Add three variable and returns the result
s = x+y+z;
```

addmult.m

```
function [s,p] = addmult(x,y)
% addmult  Compute sum and product of two matrices
s = x+y;
p = x*y;
```

函数twosum带两个输入变量，没有输出变量。它把两个输入变量的值相加并输出到屏幕上。计算结果没有赋给任何一个变量，这在MATLAB的计算中是有效的，等效于在命令窗口中定义x和y后，再输入x+y这一过程。函数twosum在NMM工具箱的program目录下面。下面的语句用于执行函数twosum（读者自己动手试试！）：

```
>> twosum(2,2)
>> x = [1 2];  y = [3 4];    twosum(x,y)
>> twosum(x,y')
>> A = [1 2; 3 4];  B = [5 6; 7 8];  twosum(A,B)
```

```
>> twosum('one','two')
```

上述函数按期望运行了吗？在MATLAB中如何计算两个字符串'one'和'two'的和？现在，再试试下面的程序：

```
>> clear
>> twosum(2,3)
>> disp([x y])
>> who
```

clear命令把用户定义的变量从工作区中清除。语句twosum(2，3)把2加3的结果输出到屏幕上。当运行函数twosum时，变量x和y分别是2和3。当程序终止时，变量x和y也不存在了（这些变量所使用的内存也被释放了）。这就是局部变量[⊖]的示例。x和y是函数twosum的局部变量，它们和工作区中其他取名为x和y的变量截然不同（和3.1.2节中的叙述进行比较）。 95

为更进一步地研究输入和输出参数，输入下列语句：

```
>> clear
>> x = 4; y = -2;            (在工作区中定义变量x和y)
>> twosum(1,2)               (在函数twosum中创建并使用本地变量x和y)
>> disp([x y])               (显示工作区的变量，不显示函数twosum定义的变量)
>> who
```

程序清单3-4中的函数threesum带有1个输出变量和3个输入变量。当threesum.m在MATLAB路径下时，输入：

```
>> a = threesum(1,2,3)
a =
    6
```

threesum的输出也可不赋给另外的变量，如：

```
>> threesum(4,5,6)
ans =
   15
```

当然，也可以把输出结果只赋给某个变量而且不输出到屏幕上，但是这要在行尾加分号，如：

```
>> b = threesum(7,8,9);
```

程序清单3-4中的函数addmult显示了如何传递多个输出参数。要注意输入参数总是用圆括号括起来，多个输出参数总是用方括号括起来。预先定义文件addmult.m并将它保存在MATLAB路径下后，可以作如下调用： 96

```
>> [a,b] = addmult(3,2)
a =
    5
b =
    6
```

当输出参数不止一个时，需要给addmult的输出结果赋值，否则，函数addmult执行的计算没有任何意义。如：

```
>> addmult(3,2)
```

结果

```
ans =
    5
```

⊖ 这时，要介绍变量作用范围的概念，这里的变量x和y的范围是在函数内部。与m文件函数中的变量相反，m文件脚本中的变量和命令窗口中变量的作用范围是相同的。

在MATLAB中，不能把返回的两个值赋给一个向量，如:

```
>> v = addmult(3,2)
v =
    5
```

97

前面的示例表明函数实际的返回参数个数应该与期望的返回参数个数相等[⊖]。

例3.4 用m文件函数绘图

MATLAB编程常用于对实验数据绘图。假设存在一个包含（x，y）数据对的纯文本文件。使用load命令把x和y的值赋给MATLAB中的变量并画出图形。但是，对不同的数据每次都要重复相同的操作，这是相当枯燥的，可以通过编写m文件函数使系统自动装载数据并绘图。

程序清单3-5中的plotData函数使用load内置函数装入数据并对这些数据创建简单图形。在函数plotData中有四条执行语句。第一条调用函数load。当数据从load返回后，把前两列元素的值赋给变量x和y，并标出（x，y）对应的点。函数清单的顶部是几行注释语句。这里使用4.1.4节介绍的格式描述函数和它的输入输出参数。

程序清单3-5 用plotData画出纯文本文件中的数据

```
function plotData(fname)
% plotData   Plot (x,y) data from columns of an external file
%
% Synopsis:  plotData(fname)
%
% Input:     fname = (string) name, including extension, of the
%                    file containing data to be plotted
%
% Output:    A plot in a separate figure window

data = load(fname);     %  load contents of file into data matrix
x = data(:,1);          %  x and y are in first two columns of data
y = data(:,2);
plot(x,y,'o');
```

在NMM工具箱的xy.dat文件中包含两列数据。要画出xy.dat中的数据，输入:

```
>> plotData ('xy.dat')
```

注意: xy.dat用单引号括起来，因为plotData和load的输入参数都是字符型变量。（可参照练习26和27中有关plotData函数的使用。）

3.2.3 主函数和子函数

在MATLAB第四版和更早的版本中，每个m文件只允许包含一个函数，5.x版本允许每个m文件函数中包含多个函数，第一个函数称为主函数（*primary function*），其他的函数称为子函数（*subfunction*），命令窗口和其他m文件中的函数只能调用主函数。同一m文件中的子函数能够互相调用。

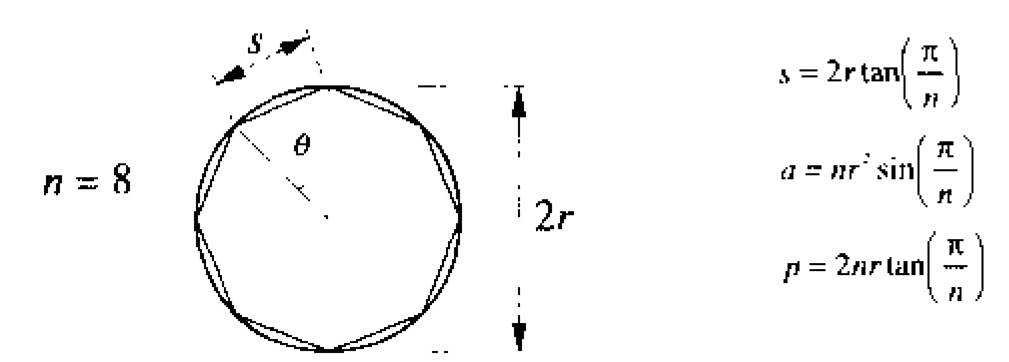
子函数有许多优点。把相似的子函数写在一个文件中，可以减少对m文件管理的复杂度。

⊖ 通过使用程序nargout创建一个函数来检测所调用函数的输出参数个数来缓解这种限制，在3.6.1节介绍输出变量个数可变的函数。

因为m文件中只有主函数能够调用子函数，所以在不同的主函数中能够使用相同的子函数名。子函数能够封装代码，使主函数更简洁易读、便于调试。当子函数多次被主函数调用时，子函数的这一优点会更加突出。

例3.5 正n边形的面积和周长 98

正n边形的面积a和周长p由下列公式计算：



$$s = 2r\tan\left(\frac{\pi}{n}\right)$$

$$a = nr^2\sin\left(\frac{\pi}{n}\right)$$

$$p = 2nr\tan\left(\frac{\pi}{n}\right)$$

程序清单3-6中的函数polyGeom调用单独的子函数计算面积a和周长p（显然，用两行MATLAB代码来计算a和p会比使用子函数调用更简洁）。命令窗口中不能调用子函数area和perimeter，这也是子函数的一个优点，因为"area"和"perimeter"比较通用，容易和其他几何图形的面积和周长的代码相混淆。

程序清单3-6 主函数和子函数的应用，用函数polyGeom计算正n边形的面积和周长

```
function [a,p] = polyGeom(s,n)
% polyGeom  Compute area and perimeter of a regular polygon
%
% Synopsis:  [a,p] = polyGeom(s,n)
%
% Input:  s = length of one side of the polygon
%         n = number of sides of the polygon
%
% Output: a = total area of the polygon
%         p = total perimeter of the polygon

r = s/(2*tan(pi/n));   %  "radius" of the polygon
a = area(r,n);
p = perimeter(r,n);

% ============ subfunction "area"
function a = area(r,n)
% area  Compute area of an n-sided polygon of radius r
a = n*r^2*sin(pi/n);

% ============ subfunction "perimeter"
function p = perimeter(r,n)
% perimeter  Compute perimeter of an n-sided polygon of radius r
p = n*2*r*tan(pi/n);
```

99

3.3 输入和输出

一般情况下，函数的输入和输出数据要通过输入和输出参数来获得，但在某些情况下，还有更好的方法来进行输入和输出。例如，在执行函数的时候提示用户输入数据。在其他情

况下，保存计算结果的格式化表格比保存没有标注的数值矩阵更实用。本节介绍支持这些输入/输出的内置函数（2.4.2节介绍了文件的输入输出）。

3.3.1 提示用户输入数据

函数input在屏幕上输出提示信息并等待用户从键盘输入数据。如:

```
>> x = input(' Enter a value for x' );
```

函数input中的参数是将要输出的提示信息。函数input的返回值默认为数值型。要使返回值为字符型数据，需要在函数input中加入第二个参数's'，如:

```
>> yourName = input(' enter your name',' s' );
```

函数input很容易被滥用。参照程序清单3-7中修改后的函数threesum，当用户多次调用函数inputAbuse时，会感到很麻烦。使用函数input输入参数会给MATLAB用户带来极大的不便。对几组不同的数据调用函数inputAbuse并分析计算结果。大部分用户在输入参数时不愿看到不必要的提示信息，特别是在大型的自动分析的函数中更是如此。最好的方法就是为输入参数提供在线帮助（参照4.1.4节），让用户通过输入参数列表输入数据。总之，使用函数input时要注意节制或干脆不用。

程序清单3-7 修改后的threesum函数滥用input函数

```
function s = inputAbuse
% inputAbuse  Use annoying input messages to compute sum of three variables
%             Compare this function with threesum

x = input('Enter the first variable to be added ');
y = input('Enter the second variable to be added ');
z = input('Enter the third variable to be added ');
s = x+y+z;
```

100

3.3.2 文本输出

MATLAB提供了两种文本输出函数：disp函数和fprintf函数。函数disp用于简单的输出任务；函数fprintf能很好地控制输出信息，也能够把结果直接输出到文件。MATLAB也提供了图形和二进制文件的输出控制（第2章介绍了绘图输出控制；可参照参考文献[75]MATLAB *Language Reference Manual*以获得二进制文件输出的详细信息）。

函数disp 函数disp只带一个变量，它可以是字符矩阵或数值矩阵。要输出简单的文本信息，只要用单引号将信息括起来:

```
>> disp(' My favorite color is red' )
```

通常，用户想给程序变量的值添加文本信息。由于函数disp只带一个变量，可以把文本信息和变量组合成一个字符串。如:

```
>> yourName = input('enter your name ','s');
>> disp(['Your name is ',yourName]);
```

函数disp的变量是字符型的行向量[' Your name is', yourName]。逗号把字符串' Your name is' 和变量yourName的值连接起来。

显示带数值变量值的文本信息时，需要用函数num2str将数值变量的类型转换成字符型，如:

```
>> x = ...                              % Assign a value to x
>> outstring = ['x = ',num2str(x)];     % A string containing the value of x
>> disp(outstring)
```

上面给显示变量outstring赋值并输出x的值，还可以直接把函数num2str的值作为函数disp的变量，如：

```
>> disp(['x = ',num2str(x)])
```

函数disp可用于打印矩阵或一组列向量中的表格数据，由于它只能带一个变量，表格中各列需要组合成一个矩阵，如下面程序所示： 101

```
>> x = 0:pi/5:2*pi;       % x is a row vector
>> y = sin(x);            % and so is y
>> disp([x' y']);         % Transpose x and y and then combine
```

x也可以为列向量，如：

```
>> x = (0:pi/5:2*pi)';    % x is a now a column vector
>> y = sin(x);            % and so is y
>> disp([x y]);           % no transpose is needed before combining
```

format命令 format命令控制数值数据在命令窗口中的显示模式，直到下一个format命令出现前，这条format命令一直有效。这里有几个示例：

```
>> format short;  disp(pi)
     3.1416
>> format long;   disp(pi)
     3.14159265358979
```

表3-1总结了format命令在数值数据显示模式中的作用。format命令会影响disp显示的数值以及不以分号结尾的表达式的值的显示格式，但函数fprintf不受format命令的影响。

表3-1 format命令对disp(x)语句的影响(x = 1.23456789)

x=1.23456789; disp(x);	
format语句	输　出
format short	1.2345
format long	1.23456789000000
format short e	1.2345e+00
format+	+
format bank	1.23

102

函数fprintf MATLAB中的fprintf函数类似于C语言中的fprintf函数，它为打印输出提供更多的控制。相应地，在语法上更复杂。函数fprintf的一般形式如下：

```
fprintf(format)
fprintf(format,variables)
fprintf(fid,format,variables)
```

这里*format*是文本的输出格式；*variables*是输出变量，其中，各变量用逗号隔开；*fid*是文件标识符，表示数据输出的目的文件，*fid*的值由函数fopen的返回值给出。

fprintf的第一种形式是用于打印简单的文本信息，如：

```
>> fprintf('Warning:  x is negative\n');
```

文本信息用单引号括起来，并用\n（换行符）结束。（C程序员注意：函数fprintf中的格式是单引号而不是双引号。）若没有\n时，输出文本信息，命令提示符紧接在"negative"

后。在函数fprintf格式行中，在输出信息之前可以使用多个换行符。当希望插入换行时，也可以将正文信息拆分到几个fprintf函数中。

函数fprintf的第二种形式是:

```
fprintf(format,variables)
```

按*format*格式输出内部变量的值。下面的例子介绍如何在屏幕上打印字符型变量和标量的值:

```
>> name = 'Elvis';   age = str2num(datestr(now,10))-1935;
>> fprintf('%s is %d years old\n',name,age);
Elvis is 65 years old
```

格式串中包括格式码和附加的字符，它们组合后产生输出字符串。格式码表示变量输出时如何组合。格式码%s表示输出格式为字符串，格式码%d表示输出格式是整数。这些格式码将二进制变量转换成输出的ASCII码。表3-2总结了格式码的基本形式。读者可以添加说明符来指定数据的长度和数值数据的精确位数。数据长度（即输出数据的总位数）是%号和输出说明符s、d、f、e或g之间的整数，当这个整数是小数时，整数部分表示浮点数整数部分的数据长度，小数部分表示浮点数小数部分的数据长度。表3-3列出了格式码的数据宽度和精度说明符。

表3-2 函数fprintf和fscanf的格式码

代 码	变换说明
%s	格式化为字符串型
%d	格式化为整数
%f	格式化为浮点数
%e	格式化为科学计数法的浮点数
%g	格式化为%f或%e的简化格式
\n	换行
\t	插入制表符（tab）

103

表3-3 函数fprintf的格式码对输出数据的影响

值	%8.4f	%12.3e	%10g	%8d
2	2.0000	2.000e+00	2	2
sqrt(2)	1.4142	1.414e+00	1.41421	1.414214e+00
sqrt(2e-11)	0.0000	4.472e-06	4.47214e-06	4.472136e-06
sqrt(2e11)	447213.5955	4.472e+05	447214	4.472136e+05

函数fprintf的第三种形式

```
fprintf (fid, format, variables)
```

将输出数据写入指定文件中。要向某个文件写入或读出数据，都要求建立文件与当前MATLAB交互环境之间的连接。参数*fid*是函数fopen的返回值，它在fprintf函数中用于连接指定文件（2.4.2节介绍了fopen函数）。下面的程序是打开myfile.dat文件并向其写入数据:

```
x = ...                                  % define contents of x vector
fout = fopen('myfile.dat','wt');         % open myfile.dat for output
fprintf(fout,'   k      x(k)\n');
for k=1:length(x)
   fprintf(fout,'%4d     %5.2f\n',k,x(k));
```

```
end
fclose(fout)                    % close myfile.dat
```

for...end循环用于遍历向量x中的所有元素。参照3.4.5节有关MATLAB循环的介绍，参照参考文献[73]*Using* MATLAB或从在线帮助中查看fscanf、fopen和fclose函数的详细信息。 [104]

3.4 流程控制

除最简单的算法外，几乎所有算法都会包含条件判断语句。许多算法中都会对同一公式反复计算，或者用同一公式对大量相似数据进行反复计算。条件语句包括if...else...及相关结构。重复语句包括for...end循环和while...end结构，这些语句在执行前都要先进行条件的真假测试。

本节首先介绍用来建立条件测试关系式的关系操作符和逻辑操作符，然后介绍if...else...结构，再介绍for...end和while...end循环，最后介绍break和return语句。

3.4.1 关系运算符

语句“is A equal to B?”用来比较参数A和B是否相等，“is equal to”是关系运算符。参数“A”和“B”可以分别是复杂表达式。表3-4列出了MATLAB中的关系运算符。

在赋值语句中可以使用关系运算符，如:

```
>> a = 2;  b = 4;
>> aIsSmaller = a < b
aIsSmaller =
     1

>> bIsSmaller = b < a
bIsSmaller =
     0
```

关系运算的结果或是“true”，或是“false”。“true”对应非零的数值数据，“false”对应零。许多计算机语言中，使用Boolean数据类型存储“true”值和“false”值。在MATLAB中，关系运算的结果可赋给矩阵变量。上面例子中的变量aIsSmaller和bIsSmaller是标量，可用于后面的数值计算中。逻辑运算可以组合关系表达式，与“and”、“or”和“not”运算符进行，如: [105]

```
>> bothTrue = aIsSmaller & bIsSmaller
bothTrue =
     0

>> eitherTrue = aIsSmaller | bIsSmaller
eitherTrue =
     1

>> ~eitherTrue
ans =
     0
```

大小相同的矩阵间可以进行关系运算，如:

```
>> x = 1:5;    y = 5:-1:1;
```

```
>> z = x>y
z =
     0     0     0     1     1
```

计算结果是包含0和1的向量（“true”和“false”值）。这一向量能够当作从矩阵中析取元素的符号表达式来使用，其相关内容将在3.5.3节中介绍。

表3-4 MATLAB中的关系运算符和逻辑运算符。在计算过程中正确区分关系运算符和逻辑运算符的优先级是很重要的（参照3.4.2节）

运算符	类型	意义	参数个数
<	关系型	小于	2
<=	关系型	小于等于	2
>	关系型	大于	2
>=	关系型	大于等于	2
==	关系型	等于	2
~=	关系型	不等于	2
&	逻辑型	与	2
|	逻辑型	或	2
~	逻辑型	非	1

106

3.4.2 运算符的优先级

在MATLAB中，有明确的规则规定关系运算符和算术运算符的运算优先顺序。如:

```
>> x = 2 + 3^2/4
```

等价于

$$x=2+\frac{3^2}{4}, \text{ 而不是 } x=2+3^{2/4},$$

因为指数运算符优先级高于除法运算符。高优先级的运算符先运算，同等级别优先级的运算符按从左到右的顺序运算。

优先级的规则有如下3条:

1. 算术运算符　　最高优先级
2. 关系运算符　　↓
3. 逻辑运算符　　最低优先级

算术运算符优先级高于关系运算符，关系运算符优先级则高于逻辑运算符。因此，表达式

```
>> y = 3>8-2 | sqrt(15)>4
```

中，先计算8-2，再与3比较，得到“false”。然后计算sqrt(15)>4，得到“false”。最后，执行逻辑或运算，结果为“false”，赋给y。

表3-5列出了算术运算符的优先级层次。优先级最高的运算符为圆括号。使用圆括号后，可使表达式更明确易懂。定义优先级让计算机能够按照明确的规则进行计算，这可称为代码的人性化，代码的人性化能够防止程序出错。如，参照如下两个等式:

```
x = s.^2/t/3*sin(pi/4*n)          x = ((s.^2)/(3*t))*sin(n*pi/4)
```

107

右边等式中附加的圆括号使公式的计算更加简明易懂。

表3-5　算术运算符的优先级。水平线把相同优先级的运算符分为一组。下面每组运算符的优先级低于上面组中的运算符。表达式中包含同组中的运算符时，按照从左到右的顺序计算

运算符	描述
.'	按元素转置（复共轭）
.^	按元素取指数
'	矩阵转置（复共轭）
^	矩阵的幂
+	一元加
-	一元减（负号）
.*	按元素相乘
./	按元素右除
\	按元素左除
*	矩阵乘
/	矩阵右除
\	矩阵左除
+	加
	减
:	冒号运算符

3.4.3 if...else语句

关系运算通常使用if...else结构有选择地执行某些代码段。一般格式如下:

```
if expression
    block of statements
end
```

当*expression*为真时，执行代码段*block of statements*。*block of statements*代码段不要求缩进，但为了便于阅读代码，建议采用缩进格式。下列例子是一个简单的if结构:

```
if a < 0
   disp('a is negative');
end
```

语句if和end没有计算表达式的值，所以不必加分号。短的if...else结构的语句可以写在同一行中，如:

```
if a < 0, disp('a is negative'); end
```

为了使MATLAB解释器能够明确地解释语句，这里需要用逗号将“if a < 0”和后面的语句隔开。

结构if...elseif...end表示从两种情况中选一种:

```
if x >= y
   c = x^2 - y;
elseif y/x > 0.0
   c = log(y/x);
end
```

当第一个条件为假且第二个条件为真时，执行elseif代码段。如，当x = -2，y = 5时，两个条件都为假，这种情况下，c将不会被赋任何值，这可能不是编程者的用意。另一种结构是if...else...end，这种结构能保证至少有一个代码段会被执行，如:

108

```
if x >= y
   c = x^2 - y;
elseif y/x > 0.0
   c = log(y/x);
else
   fprintf('WARNING: either x and y are both negative or x<y\n');
   fprintf('x = %f     y= %f\n',x,y);
end
```

要注意到最后一个选择条件是由else引出的，当前面所有的条件都不满足时，将执行else后的代码段。通常，这种默认的条件有利于复杂的逻辑测试。

3.4.4 使用switch结构进行条件选择

switch结构和if...else...结构中的elseif作用相类似。switch结构的语法如下：

```
switch  expression
  case value1
    block of statements
  case value2
    block of statements
       :
  otherwise
    block of statements
end
```

109

switch语句中的*expression*的值可为数值（通常是对表达式计算得出整数）或字符串，*value1*、*value2*...表示*expression*可能的计算结果。语句*block of statements*是有效的MATLAB语句，包括函数调用。在switch语句中，只能执行一个case语句，当*expression*与多个case匹配时，选择第一个case并执行其后的*block of statements*。结构中只有一个otherwise，当没有与*expression*匹配的case时，执行otherwise后的语句。（C程序员注意：这里的case语句后面没有break语句。）

switch结构适合对一系列离散值进行精确匹配，从而作出逻辑上的选择，因此，一般不用switch语句来匹配浮点数。（参照5.2.2节。）

下列是使用switch结构示例的一部分。

```
x = ...
switch  sign(x)
  case -1
    disp('x is negative');
  case 0
    disp('x is exactly zero');
  case 1
    disp('x is positive');
  otherwise
    disp('sign test fails');
end
```

内置函数sign返回变量x的符号值（-1，0，+1），switch代码段根据x的符号值输出相应的信息。当然，用if...else结构也可以实现相同的效果。

例3.6 在比热公式中赋常量值

零压力下CO_2、乙烷（ethane）、庚烷（heptane）、辛烷(octane)、戊烷(pentane)和其他物

质的质量定容热容可用下面实验式计算（参照文献[62]）:

$$c_v = \frac{a_1}{T} + a_2 + a_3 T + a_4 T^2 + a_5 T^3 + a_6 T^4$$

其中a_i是常量，每种化学物质所带的常量不同；T是开尔文（Kelvins）绝对温度。程序清单3-8中的函数cvcon使用switch结构根据不同的化学物质选择不同的常量进行计算。 [110]

程序清单3-8　使用函数cvcon给零压力下的各种物质所带的常量赋值

```
function a = cvcon(material)
% cvcon  Use switch construct to assign constants in curve fit for cv of fluids
%
% Synopsis:  a = cvcon(material)
%
% Input:    material = (string) name of material;  One of 'CO2','ethane',
%                      'heptane', or 'octane'
%
% Output:   a = vector of coefficients for the curve fit of zero pressure
%               specific heat of the form:
%                 cv = a(1)/T + a(2) + a(3)*T + a(4)*T^2 + a(5)*T^3 + a(6)*T^4

switch lower(material)   %  all comparisons are lower case
  case 'co2'
    a = [8726.361  184.004  1.914025  -1.667825e-3  7.30595e-7  -1.25529e-10];
  case 'ethane'
    a = [26209.109  397.31855  2.0372154  6.3813897e-3  -7.21845581e-6 ...
         2.2048025e-9];
  case 'heptane'
    a = [119252.13  -772.31363  7.4463527  -3.0888167e-3  0.0   0.0];
  case 'octane'
    a = [40859.678  -322.50398  6.6958265  -2.6759063e-3  0.0   0.0];
  otherwise
    error(sprintf('specific heat data for %s not available',material));
end
```

3.4.5 for循环

循环用来执行需要重复执行某些程序段的任务。它有两个最普通的应用：一是执行遍历向量或矩阵元素的计算；二是满足某一条件才终止的迭代计算。在MATLAB中用for...end和while...end实现循环。

for循环的一般格式如下：

```
for index = expression
    block of statements
end
```

建议读者在编程时将*block of statements*部分采用缩进格式。下面是对向量元素相加的简单循环示例[⊖]。 [111]

```
x = ...                    % create a vector
sumx = 0;                  % initialize the sum
```

⊖ 这里摘录的代码用于介绍for循环的使用而不是代码的优化（参照3.5节中用高效的向量化语句代替循环的介绍）

```
for k = 1:length(x)
   sumx = sumx + x(k);
end
```

表达式k = 1:length(x)创建具有单位增量、且与向量x有相同元素个数的行向量1:length(x)。对向量1:length(x)的每列执行一次for循环，把k的值赋给向量1:length(x)中对应的元素。

在for循环语句中，通过构造适当的向量可以创建一个任意下标的循环。如，下标步长为2的循环语句为：

```
for k = 1:2:n
   ...
end
```

下标值递减的循环语句如下：

```
for k = n:-1:1

end
```

对于非整数递增的循环语句如下所示：

```
for x = 0:pi/15:pi
   disp(sin(x));
end
```

MATLAB中的for...end语法比其他编程语言的语法更灵活。循环的次数由for语句中向量表达式右边的列数决定。对于for x = 0:pi/15:pi语句，执行16次循环，因为length(0:pi/15:pi) = 16。而且，下标变量不必是标量。计算矩阵每列的平均值的例子如下：

```
A = [1 2 3; 4 5 6; 7 8 9; 10 11 12];     % Create a matrix
for v = A                                 % Loop over columns of A
   disp(mean(v));                         % Print average of current column
end
```

[112]

因为矩阵A只有3列，所以执行3次循环，每次循环中，把矩阵A的某一列的值赋给v。仔细考虑上面的例子，假如把v = A改成v = A'将会出现什么情况？

例3.7　霍纳（Horner）法则

四阶多项式

$$p_4(x)=b_1+b_2x+b_3x^2+b_4x^3+b_5x^4=\sum_{i=1}^{5}b_ix^{i-1}$$

可用下列式子计算：

```
p=b(1)+b(2)*x+b(3)*x^2+b(4)*x^3+b(5)*x^4          (3-1)
```

假定系数b_i存储在向量b中。虽然方程（3-1）更直接（而且代数上是正确的），但并不是最好的，因为它效率不高。用霍纳法则(*Horner's rule*)，也叫嵌套相乘（*nested multiplication*），计算上式会更快，且运算量更少：

```
p=b(1)+x*(b(2)+x*(b(3)+x*(b(4)+x*b(5))));          (3-2)
```

方程（3-1）执行10次乘和4次加，而方程（3-2）执行4次乘和4次加。霍纳法则涉及很少的浮点计算，所以更高效且没有舍入误差（参照第5章）。

把多项式的系数存储在向量中，能够用for循环实现霍纳法则。

```
n = ... ;           % the polynomial is of order n-1
x = ... ;           % evaluate the polynomial at this value
p = b(n);
for k=n-1:-1:1
    p = p*x + b(k);
end
```

有时候，需要同时使用多项式和它的派生多项式。如

$$p_n(x)=\sum_{i=1}^{n+1}b_ix^{i-1} \Rightarrow \frac{\mathrm{d}p_n}{\mathrm{d}x}=\sum_{i=2}^{n+1}b_i(i-1)x^{i-2}$$

用简单的循环就能计算$p_n(x)$和$\mathrm{d}p_n/\mathrm{d}x|_x$。（参照习题20。）但是，在MATLAB中常用内置函数polyval计算多项式的值而不用更高效的霍纳法则（参照2.3.3节和习题21）。

注意内置函数计算多项式的值时，多项式按x的降幂排列（参照2.3.3节）。嵌套相乘的思想可以用在x的升幂和降幂中。 113

3.4.6 while循环

while循环的一般格式为：

```
while expression
    block of statements
end
```

建议用户编程时缩进*block of statements*。当*expression*的计算结果为“真”时，执行循环体。

和for循环相比，while循环用在循环次数不确定的计算中。它是判断初始条件并当条件满足时，执行反复操作的典型。下面是一个简单的循环（仅作参考，没有其他的用处）。

```
x = 1;
while x > 0.01
   x = x/2;
end
disp(x)
```

当第一次执行while语句时，显然x大于0.01，那么执行循环体。当测试结果为“假”时跳出循环。在这里也可用for循环代替while循环（参照习题6）。

例3.8 用牛顿迭代法计算 $\sqrt{x}$

用牛顿法计算x的开平方根的公式如下：

$$r_k=\frac{1}{2}\left(r_{k-1}+\frac{x}{r_{k-1}}\right) \tag{3-3}$$

这里r_k是$\sqrt{x}$的第k次近似值。首先，假设计算的初值r_0为x，当后面紧接着的两个近似值之差近似为零时，$\sqrt{x}$的近似值取这两个值中任意一个。这种算法的实现如下：

```
delta = ...                  % set convergence tolerance
r = ...                      % initialize r and rold
rold = ...
while abs(rold-r) > delta
   rold = r;                 % save old value for convergence test
   r = 0.5*(rold + x/rold);  % update the guess at the root
end
```

114

这里delta是所选定的收敛公差（参照5.2.4节）。由于开始进行计算时循环次数未知，所以使用while循环比for循环更好。通常，最好给while循环的循环次数确定一个上限。下面列出计算开平方根的另一种比较可靠的算法。

```
delta = ...                    % set convergence tolerance
r = ...                        % initialize r and rold
rold = ...
it = 0;                        % intialize iteration counter
maxit = 25;                    % maximum number of iterations
while abs(rold-r) > delta & it<maxit
   rold = r;                   % save old value for convergence test
   r = 0.5*(rold + x/rold);    % update the guess at the root
   it = it + 1;                % increment the iteration counter
end
```

逻辑判断要求while语句的两个条件都满足时，才执行循环体。

3.4.7 break命令

在while...end结构的循环计算中，可以使用一些跳出循环的机制以防止无限循环。通常情况下，可以设定循环次数。下面的例子在x的值小于公差时，跳出循环。

```
iter = 0;              % iteration counter
tol = ...              % convergence tolerance
xold = ...             % initial values
x = ...
while iter<maxit
   xold = x;                      % save for convergence check
   x = ...                        % update x
   iter = iter + 1
   if abs((x-xold)/xold)<tol      % Test for convergence.  If true,
     break;                       % jump to first statement after
   end                            % the enclosing while...end
end
```

遇到break命令时，跳出循环，转到while循环下面第一条语句。break命令也能用在for...end循环中（参照习题6）。注意下面的语句中：

115

```
while ...
    if ...
       break;
    end
end
```

break不是从if...end结构中跳出，而是从while...end结构中跳出，即跳出while循环。

例3.9　break命令的应用

程序清单3-9中的函数demoBreak示范了break命令的使用。长度为n的向量中的元素是0到1之间的随机数（参照"help rand"）。用循环检查向量中的每一元素。函数返回第一个大于0.8元素的下标。例如，输出结果可能是：

```
>> demoBreak(5)        %  your results are likely to differ
ans =
     3
```

```
>> demoBreak(35)
ans =
     1

>> demoBreak(35)
ans =
     7
```

注意到break命令是从while...end结构中跳出循环而不是从if...end结构中跳出。当x(k)>0.8时，程序跳转到执行while...end结构后的fprintf语句。

函数demoBreak有个小缺陷，在程序最末行的注释语句中也提到了。课后练习40的目标就是改正程序的这一缺陷。 116

程序清单3-9 break命令在函数中的应用

```
function k = demoBreak(n)
% demoBreak  Show how "break" command causes exit from a while loop
%            Search a random vector to find index of first element
%            greater than 0.8.
%
% Synopsis:  k = demoBreak(n)
%
% Input:     n = size of random vector to be generated
%
% Output:    k = first (smallest) index in x such that x(k)>0.8
x = rand(1,n);
k = 1;
while k<=n
   if x(k)>0.8
     break
   end
   k = k + 1;
end
fprintf('x(k)=%f   for k = %d   n = %d\n',x(k),k,n);

% What happens if loop terminates without finding x(k)>0.8 ?
```

3.4.8 return命令

使用break命令能终止循环，而使用return命令能终止当前正执行的函数（m文件）。遇到return命令时，当前定义的所有输出变量的值传回给调用函数。如果在命令行调用函数，则将输出变量的值传回给工作区。图3-3中描述了break语句和return语句的不同点。

例3.10 return命令的应用

程序清单3-10列出的函数demoReturn是例3.9的变体。这里没有使用break命令跳出循环，而使用了return命令跳出整个函数。函数demoReturn存在与函数demoBreak同样的小缺陷。

程序清单3-11中的函数demoArgs和程序清单3-12中的函数H2Odensity也是return命令应用的示例。

```
function k = demoBreak(n)

...

while k<=n
   if x(k)>0.8
     break;
   end
   k = k + 1;
end
```

跳转到"while...end"循环的末尾

```
function k = demoReturn(n)

...

while k<=n
   if x(k)>0.8
     return;
   end
   k = k + 1;
end
```

返回到调用函数

[117]

图3-3 break语句和return语句的不同点

程序清单3-10 return命令在函数中的应用

```
function k = demoReturn(n)
% demoReturn  Show how "return" command causes exit from a function
%             Search a random vector to find index of first element
%             greater than 0.8.
%
% Synopsis:  k = demoReturn(n)
%
% Input:     n = size of random vector to be generated
%
% Output:    k = first (smallest) index in x such that x(k)>0.8
x = rand(1,n);
k = 1;

while k<=n
   if x(k)>0.8
     return
   end
   k = k + 1;
end           %  What happens if loop terminates without finding x(k)>0.8 ?
```

3.5 向量化

向量化（*vectorization*）是指把对标量运算的代码改成对向量运算的代码。当原来的代码中包含for和while循环对矩阵或向量中的元素遍历操作时的改变更加明显。向量化不改变程序的计算次数，只是改变了MATLAB进行这些计算的效率。向量化语句执行速度比等效的标量语句更快，因为它们在MATLAB内核中由优化的二进制代码执行，而不是由m文件中解释后的标量代码执行。

在2.2.5节介绍了通过向量化编写向量运算和矩阵运算的简化代码。本节将介绍向量化在提高MATLAB代码性能上的作用。[118]

3.5.1 用向量操作代替循环

使用while和for循环结构能够编写出直接由Fortran或C转化过来的MATLAB程序。这样的程序在语法上是正确的，但大量的标量操作大大降低了代码的效率。考虑用循环计算$y =$

sin(3x)，其中，$0 < x < 2\pi$:

```
dx = pi/30;
nx = 1 + 2*pi/dx;
for i = 1:nx
   x(i) = (i-1)*dx;
   y(i) = sin(3*x(i));
end
```

以上的代码没有任何错误，但是，用这种方法创建向量x和y是相当低效的。原因有两个：主要原因是因为每次只对x(i)和y(i)中的一个元素都进行解释计算；其次，每次循环时，MATLAB需要扩展向量x和y的长度以容纳新的元素（3.5.2节介绍内存分配）。

创建向量x和y的更好的方法如下所示：

```
x = 0:pi/30:2*pi
y = sin(3*x);
```

这里不仅没有使用循环，而且代码的目的性更明确，易于阅读（需要对冒号运算符比较熟悉）。

拷贝操作 矩阵（或向量）拷贝时最好不要用循环。使用冒号运算符能够实现向量化拷贝，这样能提高计算的效率。后面会通过例子介绍冒号运算符的应用。

把矩阵中的一列拷贝到与它行数相等的矩阵的某列中。如，矩阵A和B:

```
>> A = [1 2 3; 4 5 6; 7 8 9],
>> B = ones(size(A));
```

把矩阵A的第一列拷入B的第一列。标量拷贝和与之等价的向量化拷贝分别如下所示：

119

标量代码	向量代码

`for i=1:3`
`   B(i,1) = A(i,1);`
`end`
	`B(:,1) = A(:,1);`

只要原矩阵和目的矩阵的元素个数相同，而且矩阵下标的范围是有效的，就可以对矩阵的行和列的组合进行拷贝操作。如，矩阵A的最后一列中的一部分元素可以被拷入矩阵B的第一行（或行的一部分）。标量拷贝和与之等价的向量拷贝分别如下所示：

标量代码	向量代码

`for j=2:3`
`   B(1,j) = A(j,3);`
`end`
	`B(1,2:3) = A(2:3,3)'`

使用冒号运算符进行向量化拷贝会降低代码的可读性。为了提高代码的易读性，编程者应注意在向量化拷贝时添加注释语句。

3.5.2 对向量和矩阵预分配内存

虽然MATLAB会为矩阵（或向量）自动增加内存以容纳新的元素，但是采用为矩阵（或向量）预分配内存的机制会更好些，当在循环内部每循环一次给矩阵的一个元素赋值时，采用这一机制特别重要。预分配内存是指在引用矩阵中每个单独的元素之前先用一句向量化语句创建一个矩阵（或向量）。函数ones和zeros常用于预分配内存。

假设向量s在执行循环之前没有被定义，如：

```
y = ...          % some computation to define y
for j=1:length(y)
   if y(j)>0
     s(j) = sqrt(y(j));
   else
     s(j) = 0;
   end
end
```

该循环语句没有任何错误，但是s不断地增加元素导致代码极其低效。循环之前使用s = zeros(size(y))为向量s预分配内存，如：

120

```
y = ...          % some computation to define y
s = zeros(size(y));
for j=1:length(y)
   if y(j)>0
     s(j) = sqrt(y(j));
   end
end
```

预分配内存有两条优点：首先，只分配一次内存；其次，矩阵的元素存储在MATLAB内存空间的连续块中。注意到在本例中，预分配内存还省去了循环体中s(j) = 0的操作，这也提高了代码的效率。

3.5.3 向量化索引法和逻辑函数

MATLAB使用矩阵作为基本的数据结构是为了简化代码，这种特点是其他编程语言所没有的。一个矩阵或向量可以作为另外一个矩阵的下标是矩阵运算的重要应用，这称为数组索引法(*array indexing*)。相对的技术就是逻辑索引法（*logical indexing*），它使用由0和1构成的矩阵从其他矩阵中选取所需元素。数组索引法和逻辑索引法都允许对矩阵的整体操作，因此不必使用循环，从而简化了代码，并使代码运行得更快。但是MATLAB编程的初学者阅读这种代码会比较困难。

数组索引法　请看下列语句：

```
>> x = 10:10:50                % Define a row vector
>> y = [5 3 1 2 4];            % Another vector with same shape as x
>> z = x(y)                    % Use y as index into x
z =
    50    30    10    20    40
```

x(y)是把向量y中的元素作为x的下标。与向量化语句z = x(y)等价的标量循环如下所示：

```
for i=1:length(y)
  z(i) = x(y(i));
end
```

循环格式可以避免如下两个错误的出现：（1）y元素的值必须介于1和length(x)之间；（2）length(y) ≤ length(x)。下面列出两个出现上述错误的例子：

121

```
>> y(1) = 6
y =
     6     3     1     2     4

>> x(y)
??? Index exceeds matrix dimensions.
```

```
>> y(1) = -2;
>> x(y)

Warning: Subscript indices must be integer values.
??? Index into matrix is negative or zero.  See release notes on changes to
logical indices.
```

图3-4总结了数组索引法的规则。

数组索引法或逻辑索引法C=A(B)的规则:

1. B中的元素个数不能多于A中元素的个数。
2. A和B都为矩阵时，引用矩阵中的元素等价于先将它们转换成列向量。也就是说，C=A(B)等价于
   X=A(:); Y=B(:); C=X(Y);
3. 数组索引法中，B中的元素应为A的有效的下标值。
4. 逻辑索引法中，B中的元素只能为0或1。

图3-4 数组索引法和逻辑索引法的规则

逻辑操作和逻辑索引法 关系表达式或逻辑函数的返回值为0或1。下列是两个例子:

```
>> y = pi<sqrt(10)
y =
     1
```

和

```
>> r = isreal(sqrt(-2))
r =
     0
```

表3-6只列出了MATLAB内置逻辑函数的一部分。注意这些逻辑函数的返回值。如，函数isreal总是返回标量值，isnan返回和输入矩阵同样形状的矩阵。函数any当输入为矩阵时，返回行向量；当输入为向量或标量时，返回标量。

122

表3-6 内置逻辑函数表样例

函数	描述	返回值
all(X)	X所有元素非零时为真	标量
any(X)	X的每列都有非零值时为真	标量（X为向量） 行向量（X为矩阵）
isreal(X)	X只有实数元素（没有虚数元素）时为真	标量
isstudent	运行的MATLAB为学生版本时为真	标量
isempty(X)	X为空矩阵时为真，[]	标量
isnumeric(X)	X为数值数据（不是字符串，单元或结构体）时为真	标量
isnan(X)	X的每个值都为NaN时为真	与矩阵X同形状的矩阵
isfinite(X)	X的每个值都小于Inf时为真	与矩阵X同形状的矩阵

如3.4.1节所介绍的，关系运算符的计算结果可以为逻辑值。对两个或多个匹配矩阵进行关系运算时，结果为只包含0和1的矩阵。例如[⊖]，考虑下列代码:

⊖ 每次调用rand时，F中的0和1的形式不同。

```
>> A = rand(3,3)
A =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

>> F = A>0.5        % Note: results depend on elements of A = rand(3,3)
F =
     1     0     0
     0     1     0
     1     1     1
```

123

表达式F = A > 0.5等价于下列标量代码:

```
[m,n] = size(A);
F = zeros(m,n);
for i=1:m
  for j=1:n
    if A(i,j)>0.5
      F(i,j) = 1;
    end
  end
end
```

F是一个普通的矩阵，当它的元素只有0和1时，它可用于矩阵A元素的逻辑索引。逻辑索引是利用一个索引矩阵（或向量）获取另一个矩阵（称为源矩阵）中的元素。索引矩阵只包含0和1，而且不能比源矩阵的元素多。给定矩阵A和F，A(F)是F对A的逻辑索引:

```
>> b = A(F)        % Note: results depend on elements of A = rand(3,3)
b =
    0.9501
    0.6068
    0.8913
    0.7621
    0.8214
```

表达式A(F)用矩阵F索引与F(i, j)=1相应的矩阵A中的第(i, j)个元素。b=A(F)的标量代码如下所示:

```
[m,n] = size(A);
k = 0;
for i=1:m
  for j=1:n
    if F(i,j) == 1
      k = k + 1;
      b(k) = A(i,j);
    end
  end
end
```

若要获取满足A(i, j)>0.5的元素，就不需要矩阵F。可以简单地写成下列形式:

124

```
b = A(A>0.5)
```

图3-4总结了逻辑索引法的规则。它们类似于数组索引法的规则，但是允许0索引值的出现

函数find　内置函数find可应用在许多逻辑索引法和数组索引法中。它的输入为逻辑矩

阵表达式，并返回输入变量中满足条件的元素组成的一维向量。find函数的语法如下：

```
indexVector = find(findCondition)
```

*findCondition*表达式生成包含0和1的矩阵。*indexVector*是由满足*findCondition*条件的元素对应的下标值构成的向量。若*findCondition*生成一个矩阵，find函数的计算结果是*findCondition*重构成列向量后由非零元素对应下标所构成的向量，如

```
>> A = rand(3,3)
A =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214

>> A>0.5
ans =
     1     0     0
     0     1     0
     1     1     1
>> find(A>0.5)
ans =
     1
     3
     5
     6
     9
```

已知A = rand (3, 3)，比较下列两个表达式：

```
>> b = A(A>0.5)
b =
    0.9501
    0.6068
    0.8913
    0.7621
    0.8214
```

```
>> i = find(A>0.5)
i =
     1
     3
     5
     6
     9
```

在左边，A(A>0.5)返回满足条件的元素值。在右边，find (A>0.5)返回满足条件的对应元素的下标。因此，A(find(A>0.5))等价于A(A>0.5)。

125

例3.11 把接近0的值置为0

函数GLnodes计算任意阶Gauss-Legendre积分规则的节点和权值（参照11.3.4节）。对奇数阶，其中一个节点在$x = 0$处。因为函数GLnodes返回值是一个接近0而不等于0的值。例如，对于3阶积分的结果如下所示：

```
>> [x,w] = GLnodes(3)
x =
   -0.7746
   -0.0000
    0.7746
w =
    0.5556
    0.8889
    0.5556
```

x(2)的值应该为0，但实际上是ε_m阶的数。

```
>> x(2)
ans =
  -1.1102e-16
```

假设把它的值精确为零是很重要的，显然语句x(2)= 0只对3阶积分规则有效。对于其他规则，节点为0的矩阵对应的阶数不是2。下列向量化的MATLAB语句找出了这些“小”元素并将其置为0。

```
>> x(abs(x)<100*eps) = 0
x =
   -0.7746
         0
    0.7746
```

上面例子中，因为对x所有元素都进行了条件判断，所以向量化的语句有点多余。根据积分的性质，当阶数为奇数时，x中间的元素值为0。以下示例利用了这一特点：

```
k = ...                    % Order of the quadrature rule
if rem(round(k),2)~=0      % True if k is odd, round(k) guarantees an integer
  x(1+fix(k/2)) = 0;       % Middle index is 1 + k/2 in integer math
end
```

其他情况下，无法预先知道“近零”元素的下标值。这时，需要先搜索目的矩阵的所有元素。

126

例3.12　增量搜索的向量化

增量搜索是指从有序向量x中找出所需元素下标的一种方法，如下列有序向量：

$$x(i) \le xhat \le x(i+1) \tag{3-4}$$

这里xhat是搜索的输入值。此类型操作在表格数据中插值时是必须的。10.3.2节中使用下列标量代码说明增量搜索的思想（在while循环中进行搜索）：

```
x = ...                 % A vector of values
xhat = ...              % The test value
n = length(x);
if xhat<x(1) | xhat>x(n)
   error(sprintf('Test value of %g is not in range of x',xhat));
end
i = n;                      % Search backward from n to 1
while x(i)>xhat & i>1
   i = i - 1;
end
```

用向量化操作替换while能够优化代码。用下列表达式能够替换标量while循环：

```
i = max(find(x<=xhat))
```

将上面式子的右边拆开可以看得更清楚。假设xhat = 3.7，x = 0:5。通过观察可以看出i = 4满足方程（3.4）。最内部的子表达式等价于：

```
>> xhat = 3.7;    x = 0:5;
>> x<=xhat
ans =
     1     1     1     1     0     0
```

这表明，x的前4个元素满足条件x(i) <= xhat。对于x <= xhat，用find函数得出

结果向量的下标:

```
>> find(x<=xhat)
ans =
     1     2     3     4
```

使用max可以选出满足x(i)<= xhat条件的最大的i值:

```
>> i = max( find(x<=xhat) )
i =
     4
```

127

当xhat等于x的最后一个元素值时，表达式max(find (x <= xhat))返回正确的下标。如，测试下列3条语句:

```
>> x = 0:5;    xhat = 5;
>> i = max( find(x<=xhat) )
i =
    6
```

当i=length(x)时，方程（3-4）中的x(i+1)表达式会导致下标错误，对x(i)和x(i+1)的线性插值引入了新的问题。通过第二次测试保证i值小于length (x)能够解决这一问题。通过下面两行语句完成了对整个序列的搜索:

```
>> i = max( find(x<=xhat) );        % Find largest i such that x(i)<= xhat
>> if i==length(x), i = i-1;  end   % Fix case where i=length(x)
```

或者

```
>> i = max( find(x<=xhat) );        % Find largest i such that x(i)<= xhat
>> i = min( i, length(x)-1 );       % Fix case where i=length(x)
```

向量化的增量搜索也能用其他方法来实现。

3.6 解决方法（deus ex machina）

由于计算机程序处理问题的能力越来越强，某些问题的逻辑关系处理变得比较复杂。例如，为了处理各种各样的例外情况，可能导致一堆if...end语句的出现。这时，需要重新设计代码，或者用*deus ex machina*[⊖]解决问题。MATLAB中的*deus ex machina*包括4种方法：输入输出参数个数可变、全局变量、feval函数和嵌入函数对象。这些方法都可用于解决实际的编程问题。

128

3.6.1 输入输出参数个数可变

m文件函数的输入输出参数如果能够根据需要变化，那么它将非常灵活且易于使用。标准工具箱中的许多函数都有这一特点。

例如，内置函数plot，它有多种调用形式，如:

```
plot(x,y);
plot(x,y,'+');
plot(x1,y1,x2,y2);
plot(x1,y1,'+',x2,y2,'ro');
```

[⊖] 希腊神话中，神主宰英雄。当某个地区处于灾难中时，神会降临人间并解救英雄。通俗地说，就是用于解决疑难问题的方案称为"*deus ex machina*"。按字面意思就是"机器神"。希腊神话中，*deus ex machina*是指神乘着仙鹤降临到出事地点。

如果不采用可变个数的输入参数，要么MATLAB工具箱需要包含大量相似的plot函数来满足不同的应用，要么plot函数需要一大堆的选项表，即使是最简单的绘图程序也是如此。可是使用可变输入参数的plot函数就方便得多。

用两个特殊变量可以使输入输出变量可选：即nargin和nargout，它们都是由m文件函数自动定义的。nargin的值是被调用函数的输入参数（自变量）个数，nargout的值是期望输出参数的个数。每次函数调用时，nargin和nargout的值可以不同，这时，被调用函数需要考虑到如何处理输入和打包输出。

支持输入输出参数个数可变这一特点可以扩展代码逻辑，特别是当输入参数表的变量类型不同时（如，字符型或数值型）更是如此。例如，对于由内置plot函数可接受的输入类型，考虑支持这些输入类型所必须的逻辑性。

例3.13 可变个数输入输出参数的示范

程序清单3-11的demoArgs函数演示了如何使用nargin和nargout来提供输入输出参数个数的可变。使用nargin时，在函数的开始要首先处理可选输入参数。

demoArgs只是函数的外壳函数，它打印出函数的输入参数个数以及它们的和与积。调用demoArgs函数时得到的输出参数个数就决定了将哪些计算值返回给调用程序。

若使用可选输入输出参数，函数定义（m文件的起始行）时就必须指定所有可能的输入输出参数。nargin和nargout值确定这些参数中的哪些是对函数的特定执行有用的。当用户希望返回这些可变个数输出参数时，才对它们进行定义。因此，在函数demoArgs中，输入参数的和存储在sumin中；只有nargout大于或等于1时，才把计算结果拷入out1中。

129

程序清单3-11 使用可变个数输入输出变量的函数demoArgs

```
function [out1,out2] = demoArgs(in1,in2,in3)
% demoArgs  Variable numbers of input and output parameters
%
% Synopsis:  demoArgs
%            demoArgs(in1)
%            demoArgs(in1,in2)
%            demoArgs(in1,in2,in3)
%            out1 = demoArgs(in1,in2,in3)
%            [out1,out2] = demoArgs(in1,in2,in3)
%
% Input:   in1,in2,in3 = optional dummy input arguments
%
% Output:  out1, out2 = optional dummy input arguments
%          If input arguments are provided, out1 is the sum
%          of the inputs, and out2 is the product of the inputs

if nargin == 0                              % process optional inputs
  disp('no input arguments');  return;
elseif nargin == 1
  disp('one input argument');
  sumin = in1;  prodin = in1;
elseif nargin == 2
  disp('two input arguments');
  sumin = in1+in2;  prodin = in1*in2;
elseif nargin == 3
```

```
  disp('three input arguments');
  sumin = in1+in2+in3;  prodin = in1*in2*in3;
else
  error('Too many inputs to demoArgs');
end

if nargout==0                              % process optional outputs
  return;
elseif nargout==1
  out1 = sumin;
else
  out1 = sumin;  out2 = prodin;
end
```

130

例3.14 求液态水密度的曲线拟合

可变个数输入输出参数的便利应用是为函数的使用提供一组默认值。程序清单3-12中的函数H2Odensity是个例子。

第9章给出了$\rho(T)$值的列表和处理程序，利用它可以创建下列形式的曲线拟合

$$\rho=c_1T^3+c_2T^2+c_3T+c_4$$

这里ρ是水的密度，T是温度，c_i是常系数。c_i的值由ρ和T的单位决定，再与T的范围值一起得出曲线拟合。用MATLAB程序实现上面公式可以有多种单位制。在标准环境下，不需要输入温度，使用可变个数输入变量实现这个程序并不难。

调用函数H2Odensity有如下3种形式:

```
rho = H2Odensity
rho = H2Odensity(T)
rho = H2Odensity(T,units)
```

可选的参数为：一个是输入参数T，它是要计算密度的当前温度值；另一个是units，它是T和输出变量rho所使用的单位。

不带输入参数时，用函数H2Odensity计算的是水温为20摄氏度且用国际公制单位时液体水的密度：

```
>> r = H2Odensity
r =
  998.2000
```

带一个输入参数时，计算的是液体水的温度为其他值时的密度:

```
>> r = H2Odensity(31)
r =
  995.3755
```

带两个输入参数时，可用华氏温度，这时返回的密度值单位为lbm/ft³:

```
>> r = H2Odensity(68,'f')
r =
   62.3139
```

注意在使用英制单位时，要特别地指明T和units。

131

程序清单3-12 函数H2Odensity利用可变输入参数个数提供缺省值并且使输入参数类型可改变

```
function rho = H2Odensity(T,units)
% H2Odensity  Density of saturated liquid water
%
% Synopsis:   rho = H2Odensity
%             rho = H2Odensity(T)
%             rho = H2Odensity(T,units)
%
% Input:   T     = (optional) temperature at which density is evaluated
%                  Default: T = 20C. If units='F' then T is degrees F
%          units = (optional) units for input temperature, Default = 'C'
%                  units = 'C' for Celsius, units = 'F' for Fahrenheit
%
% Output:  rho = density, kg/m^3 if units = 'C', or lbm/ft^3 if units = 'F'

% Notes:  Use 4th order polynomial curve fit of data in Table B.2
%         (Appendix B) of "Fundamentals of Fluid Mechanics",
%         B.R. Munson, et al., 2nd edition, 1994, Wiley and Sons, NY

if nargin<1
  rho = 998.2;  return;     %  Density at 20 C w/out evaluating curve fit
elseif nargin==1
  units='C';                %  Default units are C
end

% --- Convert to degrees C, if necessary
if upper(units)=='F'
  Tin = (T-32)*5/9;         %  Convert F to C; don't change input variable
elseif upper(units) == 'C'
  Tin = T;
else
  error(sprintf('units = ''%s'' not allowed in H2Odensity',units));
end

% --- Make sure temperature is within range of curve fit
if Tin<0 | Tin>100
  error(sprintf('T = %f (C) is out of range for density curve fits',Tin));
end

% --- Curve fit coefficients
c = [ 1.5439082497803814416e-05  -5.8780053950300498520e-03 ...
      1.7884472119458597740e-02  1.0000099267813384360e+03];

rho = polyval(c,Tin);       % Evaluate polynomial curve fit
if upper(units)=='F'
  rho = rho*6.243e-2;       % Convert kg/m^3 to lbm/ft^3
end
```

132

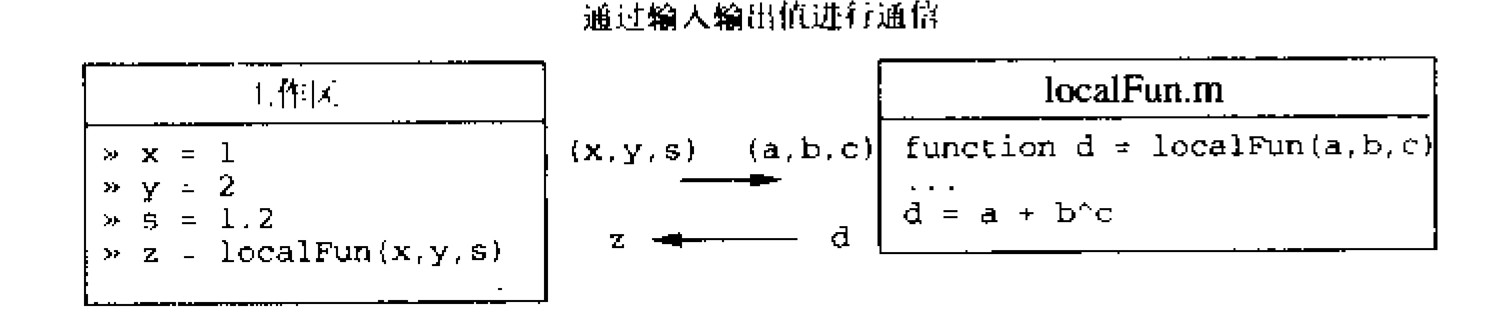
3.6.2 全局变量

使用全局变量就可以不用函数的输入输出参数列表。编程时使用全局变量能够解决高级的编程问题，但是当可以使用标准输入输出参数列表时，不要用全局变量。一些投机取巧的

程序员常常滥用全局变量，而不使用输入输出参数与m文件函数直接通信，这通常会产生一个错综复杂的m文件网，要读懂共享全局变量的m文件集同时需要读懂所有的m文件。相反，把函数之间的通信限制在输入输出参数列表中会便于读者阅读，可以把各个m文件看成单独的模块。但是，使用全局变量编程有时会带来很大的方便。

不使用全局变量时，工作区的变量（参照2.4.1节）和被调用函数的内部变量是截然不同的，图3-5的上半部分描述了这一特点。命令窗口中定义的x、y、s的值分别被拷入函数localFun的内部变量a、b、c中。

通过输入输出值进行通信



通过输入输出值和由工作区和函数共享的全局变量进行通信

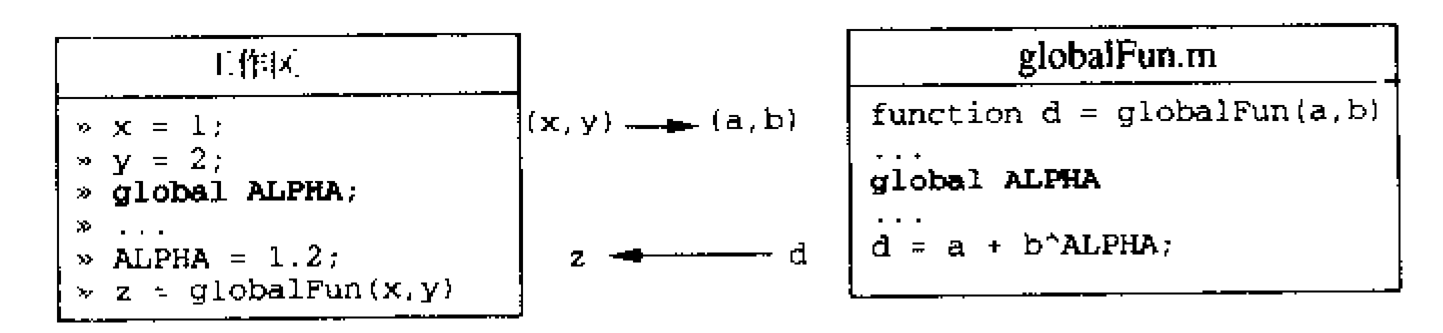


图3-5 输入输出参数与全局变量

133

计算函数localFun内的变量d的值并返回给命令窗口中的变量z。变量x、y、s和z只存在于工作区，函数localFun运行时才给变量a、b、c和d分配独立的内存空间。

全局变量能由工作区或函数直接调用。图3.5的下半部分列出了示例。和上半部分一样，x、y在命令窗口中定义，并作为输入参数拷入函数globalFun中。另外定义的全局变量ALPHA如下所示：

```
global ALPHA
```

为了突出，在本书中全局变量都用大写字母。在一个语句中也能定义多个全局变量，如：

```
global ALPHA SIGMA ZETA
```

注意，每个全局变量之间要用空格隔开，而且语句末尾不加分号。

如图3-5所示，在工作区和函数中都要定义所使用到的全局变量，而且这些变量在工作区和函数中名字要保证相同。本例中，不必声明ALPHA为全局变量，它最好作为输入参数传递给函数globalFun。

函数之间全局变量的共享和工作区与函数共享全局变量的方式一样。

3.6.3 函数feval

用户可以用多种数值算法计算实际问题。例如，计算下列式子的近似值

$$I=\int_a^b f(x)\mathrm{d}x$$

函数$f(x)$必须对区间$a<x<b$中的任意点求值。数值近似积分的通用程序可与任何有意义的$f(x)$一起操作，但是也出现了一个问题，因为程序的输入参数不是一个存储在变量x中的数值，而是用于求$f(x)$对任意x积分的程序的名字（即m文件的名字）。在MATLAB中，函数feval能够用于解决这类问题。

134

函数feval的语法如下所示：

```
feval(fun, p1, p2, ...
```

第一个参数*fun*是字符串，表示调用函数的名字。其余的参数*p1*、*p2*、...传递给由*fun*所访问的函数。如下两条语句等价：

```
>> feval('sin',pi/2);
ans =
     1

>> sin(pi/2);
ans =
     1
```

当然对上面的例子来说，没有必要用feval函数计算角度的正弦值。前面的例子只是为了说明一点：函数的直接调用等价于使用函数feval对此函数的间接调用。实际使用中，只有当一个m文件作为另一个m文件的输入参数时才调用feval函数。

例3.15　计算抽样函数的平均值

将区间$a<x<b$等分成n段，对每个等分点分别计算其函数值并求出所有函数值的和。假设s为和

$$s=\sum_{i=1}^{n}f(x_i)$$

其中

$$x_i=a+(i-1)h,\qquad h=\frac{b-a}{n-1}$$

编写一段代码专门计算对于特定函数的s值，比如说计算sin(x)的s值，这是较容易的。但这缺乏通用性，因为在计算cos(x)时还要再设计一段专用的代码。函数feval提供了更一般的解决方法，它的实现如程序清单3-13中的函数fsum。函数fsum的第一个输入参数为字符型，即函数名。例如，用fsum计算sin和cos函数在等分区间$0<\theta<\pi$为5段时，在每个等分点上的函数值的和，可输入下列语句：

```
>> fsum('sin',0,pi,5)
ans =
    2.4142

>> fsum('cos',0,pi,5)
ans =
     0
```

135

函数fsum的第一个变量必须用单引号括起来，以表示它是字符串。函数fsum能和任意函数一起使用，这要求在m文件中包含这些函数，而且这个m文件输入参数为一个向量并返回

函数值构成的向量。如，程序清单3-14中的sincos函数计算sin(x)cos(x)的积。当在MATLAB路径下创建了sincos.m文件后，下面程序

```
>> fsum('sincos',0,pi,15)
ans =
   3.3307e-16
```

程序清单3-13 用函数fsum计算用户定义间隔点上函数值的和

```
function s = fsum(fun,a,b,n)
% fsum  Computes sum of f(x) values at n points in  a <= x <= b
%
% Synopsis:   s = fsum(fun,a,b,n)
%
% Input:  fun = (string) name of the function, f(x), to be evaluated
%         a,b = endpoints of the interval
%         n   = number of points in the interval
%
% Output:  s = sum of f(x) at n discrete points in the interval

x = linspace(a,b,n);       %  create points in the interval
y = feval(fun,x);          %  evaluate function at sample points
s = sum(y);                %  compute the sum
```

就可以把区间$0 \leqslant x \leqslant \pi$等分15段后，计算在每个等分点上的函数值的和。

程序清单3-14 使用程序清单3-13中的函数fsum计算sincos

```
function y = sincos(x)
% sincos  Evaluates sin(x)*cos(x) for any input x
%
% Synopsis:  y = sincos(x)
%
% Input:      x = angle in radians, or vector of angles in radians
%
% Output:     y = value of sin(x)*cos(x) for each element in x

y = sin(x).*cos(x);
```

136

3.6.4 嵌入函数对象

MATLAB第五版增加了支持面向对象的编程。面向对象编程使用多种技术把对象和它的行为连接起来，对象是对变量的抽象扩展。对象可用于存储各种类型的实体：数、字符串、矩阵、函数、其他对象和它们的组合等。每个对象定义了许多方法，方法是函数的扩展。面向对象编程方法是组织计算任务的一种功能强大的典范，要用好这种方法需要比较高的技术水平。对于基本的数值分析工作，不需要使用面向对象的方法，但是，在复杂的程序开发中它是很有用的。

面向对象的特点之一是用嵌入函数对象（*inline function object*），这一特点可以使用户很方便地处理数值问题。定义和使用嵌入函数都很容易，这样就不必创建短m文件函数来计算简单的公式。嵌入函数对象是指计算MATLAB表达式的对象。通常，这种表达式中有1个或多个变量，类似于m文件函数的输入参数。如下例：

```
>> scfun = inline('sin(x).*cos(x)')
scfun =
     Inline function:
     scfun(x) = sin(x).*cos(x)
```

语句scfun = inline(' sin(x).*cos(x)')创建了scfun嵌入函数对象。系统对该语句的响应如下:

```
Inline function:
scfun(x) = sin(x).*cos(x)
```

表明MATLAB把x作为函数对象scfun的输入变量。MATLAB语法分析程序自动识别嵌入函数对象中的变量。使用help inline能够看到有关系统如何识别这些变量的更多信息。

定义scfun后，就可以像使用m文件函数一样使用它。也就是说，可以用函数scfun代替程序清单3-14中的sincos函数:

```
>> scfun(5)-sin(5)*cos(5)
ans =
     0

>> scfun(0:pi/4:pi)
ans =
         0    0.5000         0   -0.5000         0
```

137

嵌入函数对象和m文件函数有很大的区别，嵌入函数对象是在MATLAB工作区中创建的，它只存在于当前的交互环境中:

```
>> whos
  Name          Size           Bytes  Class

  ans           1x5               40  double array
  scfun         1x1              844  inline object

Grand total is 52 elements using 884 bytes
```

当嵌入函数对象被赋予某个MATLAB变量时，它可以作为参数传给另一个函数。例如:

```
>> fsum(scfun,0,pi,15)
ans =
   3.3307e-16
```

把scfun对象传给例3.15中的函数fsum。注意函数scfun没有用单引号括起来，和下列式子进行比较:

```
>> fsum('sincos',0,pi,15)
ans =
   3.3307e-16
```

它们的计算结果一样。

在m文件函数中也能创建嵌入函数对象，本书通篇都使用了这一特点。程序清单3-15显示了如何在m文件中创建嵌入函数对象。

程序清单3-15 函数demoXcosx示范了在m文件函数中如何定义和使用嵌入函数对象

```
function s = demoXcosx(n)
% demoXcosx  Use an inline function object with the fsum function
%
```

```
% Synopsis:  s = demoXcosx
%
% Input:     n = number of points at which x*cos(x) is evaluated
%                in the interval 0 <= x <= 2*pi
%
% Output:    s = sum of x*cos(x) at n points

xcosx = inline('x.*cos(x)');
s = fsum(xcosx,0,3*pi,n);
```

138

3.7 小结

本章介绍了MATLAB中程序模块的创建，介绍了m文件脚本和m文件函数。由于函数常用于结构化编程，所以推荐读者使用m文件函数。

本章的主要目的是描述和示范MATLAB语言的结构。所举的例子都是以代码段和m文件的形式提供的。表3-7列出了本章介绍的m文件函数和m文件脚本。它与后面章节中介绍的m文件不同，在表3-7列出中的m文件主要是为本章介绍专门的编程技术而设计的，而不是为了在其他工程上应用它们。

补充读物

MATLAB编程的文献来源主要是当前版本MATLAB的参考手册。help和helpwin在线帮助为用户提供了每个函数的简明概述。每当新的MATLAB版本发布时，都会附带印刷版手册，以及提供了更详细信息的PDF格式的电子手册。想知道高级MATLAB命令信息的读者还可以与Hanselman和Littlefield联系，请参考文献[34]。

表3-7 介绍编程概念的NMM工具箱函数

函 数	小 节	描 述
addmult	3.2	计算两个矩阵的和与积
demoArgs	3.6	可变个数输入输出参数的示例
demoBreak	3.4	break语句的示例
demoReturn	3.4	return语句的示例
demoXcosx	3.6	在m文件函数中如何定义和使用嵌入函数对象的示例
fsum	3.6	计算n等分区间$a \le x \le b$在各等分点的$f(x)$值的和
H2Odensity	3.6	计算饱和液态水的密度
inputAbuse	3.3	用户每次输入一个数值，计算三次输入的和。不要使用input函数的示例。与函数threesum作比较
myCon	3.1	在工作区定义有用的常量
plotData	3.2	对纯文本文件中的数据绘图
polyGeom	3.2	主函数和子函数使用的示例
sincos	3.6	对任意x计算$\sin(x)\cos(x)$
takeout	3.1	显示餐馆电话的脚本
threesum	3.2	计算3个矩阵的和并返回结果
trigplot	3.1	画$\sin(x)$，$\cos(x)$和$\sin(x)\cos(x)$对应的图形
twosum	3.2	计算2个矩阵的和并打印结果

139

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量（参照1.4节中关

于分级系统的介绍)。

1. (1)根据自己的需要更改myCon脚本程序，使它包含所需常量和转换因子。

2. (1　)修改程序清单3-4中的函数twosum，将它的返回值赋给变量s，不允许在函数twosum内部打印结果。

3. *(1+)把程序清单3-2中的takeout脚本程序变换成函数，并添加一个输入变量kind来选择自己喜欢的餐馆。在函数体内添加多重选择if...elseif...end结构来匹配变量kind。若kind不与任何餐馆匹配，一定要有默认选项——餐馆类型或错误信息。(提示：可通过使用函数strcmp比较字符串是否相等来实现。)

4. (1+)优化上题中的函数takeout。不比较kind与餐馆是否精确匹配，只要比较前n个字符，这n个字符的长度足以区分餐馆的类别。再创建一个内部变量lowerkind，它与kind中的字符相同，但都是小写的。比较lowerkind和餐馆类别名为小写的情况。这样会使比较操作对大小写不那么敏感。

5. (1)使用牛顿算法写一个newtsqrt函数来计算某个数的开平方根。所编程序必须包含3.4.6节最后那段代码。

6. (1+)使用牛顿算法写一个newtsqrt函数来计算某个数的开平方根，要求用for...end结构的循环。在循环数没有达到最大值，但是收敛性条件满足时，该如何终止循环?

7. *(1)Elvis Presley生于1935年1月8日，死于1977年8月16日，假设他还活着，编写函数elvisAge计算并返回Elvis Presley按年份计算的实际年龄。(参照3.3.2节中相关代码。)

8. (2)同上题，要求添加输入变量onDate。返回Elvis在onDate时的年龄，这里onDate是MATLAB中的日期(参照help　datenum)。要求所编程序能够正确计算下列测试日期时的年龄：

140

(a) 当他死于1977年8月16日(测试语句应该是elvisAge(datenum(1977, 8, 16)))。

(b) 他第一个生日的前一天：1936年1月7日。

(c) 他的第一个生日：1936年1月8日。

(提示：可以使用内置函数datevec来解题)答案：a. 42岁 b. 0岁 c. 1岁。

9. (2)同上题，扩展函数elvisAge使它能够可选择地返回从Elvis出生时算起所度过的月数与天数。调用扩展后的elvisAge函数应该返回下列值：

```
>> y = elvisAge(datenum(1977,8,16))
y =
    42

>> [y,m] = elvisAge(datenum(1977,8,16))
y =
    42
m =
     7

>> [y,m,d] = elvisAge(datenum(1977,8,16))
y =
    42
m =
     7
d =
     8
```

10. *(2)编写函数mydiag模拟内置函数diag的行为。最低要求函数mydiag应该能够对向量输入返回对角矩阵，对矩阵输入返回对角线元素(diagonal entry)。(提示：可借助函数size。)

11. (2)创建m文件函数sequint，要求它返回矩阵，且矩阵的元素为按列排序的一系列整数值。函数包括两个输入变量：m为行数、n为列数。语句sequint(2，3)和sequint(4，4)分别生成下列两个矩阵

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} \qquad \begin{bmatrix} 1 & 5 & 9 & 13 \\ 2 & 6 & 10 & 14 \\ 3 & 7 & 11 & 15 \\ 4 & 8 & 12 & 16 \end{bmatrix}$$

(提示：先创建向量，然后把向量转换成矩阵。)

12. (2　)编写两个m文件函数FtoC和CtoF，要求分别把华氏温度转换成摄氏温度和把摄氏温度转换成华氏温度。每个函数都有一个输入参数和一个输出参数。用下列语句测试所编函数

141

```
>> FtoC( CtoF(100) )
>> CtoF( FtoC(32) )
>> FtoC(0:10:100)
>> CtoF(0:10:100)
```

或用自己认为合适的测试组合。

13. (2)给定任意长度的向量x，编写一段程序，要求在同一行打印出x=，后面是向量x中的所有元素值。如，创建向量x = 0:0.25:2，要求代码产生

```
x =   0.00  0.25  0.50  0.75  1.00  1.25  1.50  1.75  2.00
```

不改变代码，当输入x = 0:0.25:1时，产生

```
x =   0.00  0.25  0.50  0.75  1.00
```

(提示：将打印操作分到多个fprintf函数中。)

14. (2)使用内置函数diag编写程序创建$n \times n$的对称三对角矩阵

$$D = \begin{bmatrix} c & d & 0 & 0 & \cdots & 0 \\ d & c & d & 0 & & \\ 0 & d & c & d & & 0 \\ & 0 & \ddots & \ddots & \ddots & \\ & & & & d & c \end{bmatrix}$$

这里c和d是标量。函数可能的一种定义形式如下function A = tridiag (c, d, n)，这里n是矩阵的维数。(提示：注意diag函数带两个输入参数的形式。)

15. (3)同上题，要求c和d可以是向量。即，若c和d为向量，创建下列矩阵

$$D = \begin{bmatrix} c_1 & d_1 & 0 & 0 & \cdots & 0 \\ d_1 & c_2 & d_2 & 0 & & \\ 0 & d_2 & c_3 & d_3 & & 0 \\ & 0 & \ddots & \ddots & \ddots & \\ & & & & d_{n-1} & c_n \end{bmatrix}$$

若c和d为向量，用上题程序创建矩阵。注意：D为对称矩阵，即$D = D^T$。(提示：通过测试c和d的长度判断输入值是标量还是向量。)

[142]

16. *(1)用循环结构和fprintf函数产生下表(数值数据的格式应与下表完全对应)。

```
theta   sin(theta)   cos(theta)
  0       0.0000       1.0000
 60       0.8660       0.5000
120       0.8660      -0.5000
180      -0.0000      -1.0000
240      -0.8660      -0.5000
300      -0.8660       0.5000
360       0.0000       1.0000
```

17. (1) 用循环结构和fprintf函数产生下列输出：

```
(a) 1^2 = 1
    2^2 = 4
    3^2 = 9
    4^2 = 16
    5^2 = 25
(b) (0.10)^2 =    0.0100
    (0.20)^2 =    0.0400
    (0.30)^2 =    0.0900
    (0.40)^2 =    0.1600
    (0.50)^2 =    0.2500
```

18. (1+)根据霍纳法则编写函数horner，计算任意次数的多项式的值。(参照例3.7)。使用如下函数定义：

```
function p = horner(b, x)
```

其中b是多项式的系数向量，返回值p是多项式在点x处的值。使用x = 1，2，3，4，5，系数向量b = [1 2 -1 0]测试所编函数。

19. (2+)同上题，要求输入x为向量时，返回值也为向量。

20. (2+)同上题，要求函数能按以下两种形式调用：

```
p = horner(b, x)
```

和

```
[p,pp] = horner(b, x)
```

这里p是多项式在点x处的值，pp（p的一阶导数，$p' = dp/dx$）是$p(x)$在点x处的导数值。注意当x为向量时，要求p和pp都以向量形式输出。

21. (1+)内置函数polyval如何计算标量多项式的值？这样做会产生什么效果？（提示：阅读函数polyval的源代码。）

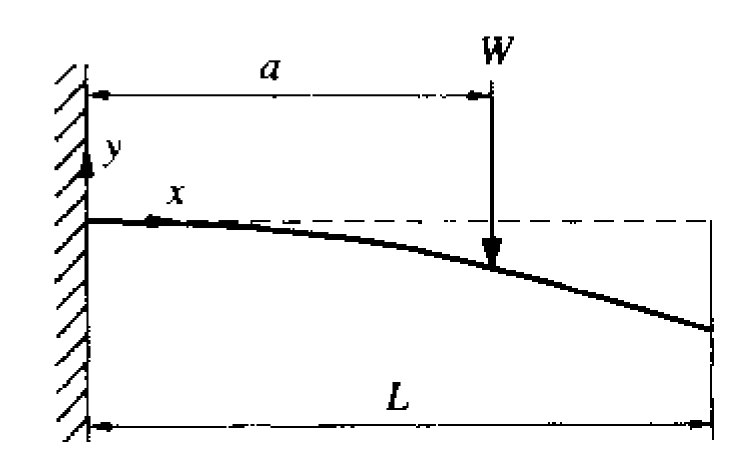
[143]

22. *(2+)编写函数画出悬臂梁在标准负载下的位移图，并返回最大位移以及悬梁曲面末端处切平面与水平面的夹角。位移v和夹角θ的计算公式如下所示：

$$y = -\frac{wx^2}{24EI}(6L^2 - 4Lx + x^2) \quad 且 \quad \theta = \frac{1}{6}\frac{wL^3}{EI}$$

这里w是每单位长度的负载，E是悬梁的杨氏模量，I是悬梁的转动惯量，L是悬梁的长度。用如下数据测试所编函数：E= 30Mpsi，I = 0.163in^4，L = 10in，w= 100 lb_f/in。

23. (2+)编写函数画出悬臂梁在点负载下的位移图，并返回最大位移以及悬梁曲面末端处切平面与水平面的夹角。梁的几何图形如下所示:



位移y和夹角θ的计算公式如下所示:

$$y=-\frac{Wx^2}{6EI}(3a-x),\quad \text{for } 0\leqslant x\leqslant a$$

$$y=-\frac{Wa^2}{6EI}(3x-a),\quad \text{for } a\leqslant x\leqslant L$$

且

$$\theta=\frac{Wa^2}{2EI}$$

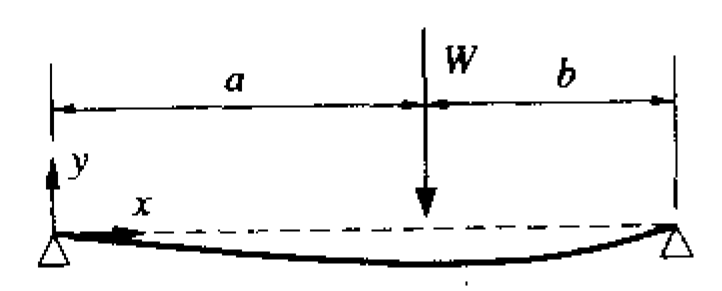
其中W是点负载，E是悬梁的杨氏模量，I是悬梁的转动惯量，L是悬梁的长度。用如下数据测试所编函数: E =30Mpsi，I= 0.163in^4，L= 10in，a = 3in，W= 1000 lb$_f$。

24. (2+)在某点的负载下支撑梁的位置和位移最大值分别是

$$x_{max}=\sqrt{a(a+2b)/3}\ \text{和}\ y_{max}=-\frac{Wab}{27EIL}(a+2b)\sqrt{3a(a+2b)}$$

这里W是负载，E是悬梁的杨氏模量，I是悬梁的转动惯量，$L=a+b$是悬梁的长度。若$a<b$，那么$x_{max}=L-\sqrt{b(b+2a)/3}$。计算$y_{max}$的公式与上式类似，只是$b$与$a$的角色正好互换。

144



对于给定W、a、b、E和I值，编写函数计算并返回x_{max}和y_{max}的值。对E=30Mpsi，I = 0.163in^4，L = 10in，W = 1000 lb$_f$，画出x_{max}和y_{max}在$0\leqslant a/L\leqslant 1$时的图形。使用内置函数`plotyy`区分$x_{max}$和$y_{max}$的图形，或者用两个单独的绘图窗口分别绘图。

25. (2)给`xy.dat`中的数据加入干扰$t=\pi(\sin(x)+\cos(x))$，其中$0\leqslant x\leqslant 5\pi/2$。(即: $y=t+$ noise)。创建函数`plotData`画出数据集(x, y)和(x, t)的图形，并画出y的平均值$\bar{y}$对应的水平线。用空心圆标出数据点(x, y)，用实线连接数据点(x, t)，并用虚线画出$\bar{y}$。使用函数`legend`标注3个图形。

26. (1)修改`plotData`函数使它画出第一行包含x，第二行包含y的数据文件的图形。使用NMM工具箱中的`xtyt.dat`文件测试所编程序。要求修改后的函数对`xtyt.dat`所

绘图形与函数修改之前对xy.dat所绘图形相同。

27. (2)修改plotData函数，使它能画出包含任意列y数据的文件所对应的图形。文件格式为

$$\begin{matrix} x & y_1 & y_2 & \cdots, \\ \vdots & \vdots & \vdots & \vdots \end{matrix}$$

这里y1，y2...对应相同的横坐标(x)。要求函数不能修改NMM工具箱data目录下的xy2.dat和xy5.dat文件（提示：可借助hold on命令）。

28. *(3)NMM工具箱data目录下的corvRain.dat文件包含俄勒冈州哥瓦里斯从1890年到1994年的降水量数据。数据按列保存，第一列为年份，后面12列是降水量，单位是百分之一英寸/每月。编写MATLAB函数计算并画出从1890到1994年，每年的总降水量(单位为英寸而不是百分之一英寸)，并打印出对应年份的平均降水量、最低降水量和最大降水量。可用NMM工具箱中的utils目录下的loadColData函数读出数据，而不必从数据中删除列标题。能否不用循环来计算每年的总降水量？

145

29. (2)同上题，计算并打印出CorvRain.dat文件中每月的平均降水量。计算并打印出从1890年到1994年每个月的总降水量。单位用英寸而不是百分之一英寸。

30. (1+)手工计算下列表达式，然后用MATLAB进行验证:

(a) `5 | 4`
(b) `~3`
(c) `x = log10(25*2^2)`
(d) `y = 5+2 & ~pi<eps`
(e) i. `s = (x>y) & (x>4)`
　　ii. `t = ~(x|y)`
　　iii. `u = x((x>y) & (x>4))`
　　已知 `x = [0 5 3 7]` 且 `y = [0 2 8 7]`.
(f) `A = reshape(1:8,2,4); B = A(5*ones(2,2))`
(g) `A = reshape(1:8,4,2); B = A(5*ones(2,2))`

31. *(2-)编写函数maxi返回向量中最大正数对应的下标。即imax = maxi(v)返回的imax满足：对所有的i值有v(imax) ≥ v(i)。

32. (2)同上题，要求只用一行代码实现(提示：使用内置find函数)。

33. (2-)编写函数maxai返回向量中绝对值最大数对应的下标。即imax = maxai (v)返回的imax满足：对所有的i值有abs(v(imax)) ≥ abs(v(i)).

34. (2)同上题，要求只用一行代码实现(提示：使用内置find函数)。

35. (2)给定向量x

```
>> x = [21 22  23 24];
```

和矩阵B

```
>> B = ones(3, 3);
```

146

用一行代码实现与下列循环等价的向量化拷贝操作:

```
k = 0;
for i=2:3
  for j=1:2
     k = k + 1;
     B(i,j) = x(k);
```

```
    end
end
```

(提示：借助于内置函数reshape。)

36. (2+)3.5.2节末的代码计算了

$$s_i = \begin{cases} \sqrt{y_i}, & 如果\ y_i > 0 \\ 0, & 其他 \end{cases}$$

写出这段代码的向量化版本(不包含循环)(提示：预分配s并使用函数find)。

37. (2+)同上题，要求预分配s后，通过对单个表达式的左边和右边分别使用逻辑索引法来解决该问题。

38. (2)根据下列定义编写函数indexCheck

```
function ok = indexCheck(A,B)
```

来验证根据图3-4中列出的规则，A(B)是否是有效的数组索引法表达式或逻辑索引法表达式。若A(B)无效，返回ok = 0并打印出适当的错误信息。否则，返回ok =1。

39. (3)编写满足下列要求的函数evenChecker：

```
function A = evenChecker(m,n)
% evenChecker  Create a checkerboard matrix of ones and zeros
%
% Syntax:  A = evenChecker(m,n)
%
% Input:   m,n = number of rows and columns in matrix
%
% Output:  A = an m-by-n matrix having ones in all elements with even
%              values of i+j and zeros in all elements with odd values
%              of i+j, where i and j are the row and column indices,
%              respectively
```

40. (2)指出程序清单3-9中函数demoBreak的缺陷，并加以纠正。要求改正后仍使用break函数。

41. (2)指出程序清单3-10中函数demoReturn的缺陷，并加以纠正。要求改正后仍使用return函数。

147

42. (1+)要求用嵌入函数对象和fsum函数计算(参照程序清单3-13)

$$s = \sum_{i=1}^{n} \tan(x)\cos(x)$$

x = linspace (0, 2*pi, n)且$n = 5$、10、25、50。

43. *(2+)编写m文件函数计算圆柱体的横向(cross flow)热传导系数。根据无量纲的努塞尔特数（Nusselt number）$Nu = hd / k$定义热传导系数，其中d是圆柱体的直径，k是流体的热传导率。用雷诺数Re（Reynolds number）和普朗特数Pr（Prandtl number）修正Nu

$$Nu = C\,Re^{m}\,Pr^{1/3}，这里\ Re = \frac{\rho V d}{\mu} \quad and\ Pr = \frac{\mu c_p}{k}$$

其中ρ, μ, c_p分别是液体的密度，动态粘度和比热，V是圆柱体附近流体的速率。参数C和m的值由下表决定：

Re范围	C	m
0.4 – 4	0.989	0.330
4 – 40	0.911	0.385
40 – 4000	0.683	0.466
4000 – 40 000	0.193	0.618
40 000 – 400 000	0.027	0.805

m文件的首行如下:

```
function h = cylhtc(d,v)
```

这里d是圆柱体的直径，v是流体的速率。处于一个大气压、20°C的环境中:

$$\rho=1.204\text{kg/m}^3,\ \mu=1.82\times10^{-5}\text{N.s/m}^2,\ c_p=1\,007\text{J/(kg K)}$$
$$k=26.3\times10^{-3}\text{W/(m°C)}$$

44. (2+)解释程序清单3-11中函数demoArgs定义的下列两条语句移到第一个end语句（第30行之后）后面将会出现什么情况:

```
sumin = in1+in2+in3;
prodin = in1*in2*in3;
```

（提示：作上述改变后在命令窗口输入demoArgs(1、3)。）

45. (2)扩展程序清单3-12中的H2Odensity函数，要求返回第二个可选字符串变量units，且返回变量不与内部变量"units"重名。

148

46. (2+)扩展练习43，要求函数cylhtc带第三个可选输入变量fluid。即m文件的第一行为:

```
function h = cylhtc(d,v,fluid)
```

以使用户可以使用下列语句中的任何一种调用形式:

```
>> h = cylhtc(d,v)
>> h = cylhtc(d,v,'air')
>> h = cylhtc(d,v,'water')
```

使用nargin变量检测cylhtc有几个输入变量。变量“fluid”是等于“air”或“water”的字符型变量(用strcmp比较两个字符串)。根据fluid的值确定计算中是使用空气还是水的Nu、Re、Pr属性值。

47. (3)使用程序清单3-12中的函数H2Odensity构造函数airProps，要求函数airProps返回密度空气密度ρ、速率μ、常压下的比热c_p和热传导率k，它们都是气压和温度的函数。函数定义语句如下所示:

```
function [rho,cp,mu,k] = airProps(T,p,units)
```

其中T是温度、p是气压、units表明采用何种单位制，它们的值都是可选的。要求函数以下列形式调用:

```
[rho,cp,mu,k] = airProps
[rho,cp,mu,k] = airProps(T)
[rho,cp,mu,k] = airProps(T,p)
[rho,cp,mu,k] = airProps(T,p,units)
```

并提供合理的T、p、units的默认值。用下列方程计算密度和热物理性质:

$$\rho = \frac{p}{RT} \qquad c_p = \sum_{i=1}^{n+1} c_{p,i} T^{n-i+1}$$

$$\mu = \sum_{i=1}^{n+1} c_{\mu,i} T^{n-i+1} \qquad k = \sum_{i=1}^{n+1} c_{k,i} T^{n-i+1}$$

这里p是空气的压强、R = 287.0J/kg/K是空气的理想气体常数、T是开尔文绝对温度($T(K)$=273.15+T(°C))、$c_{p,i}$、$c_{\mu,i}$和$c_{k,i}$是下表中列出的常量

i	$c_{p,i}$	$c_{\mu,i}$	$c_{k,i}$
1	$2.455322455 \times 10^{-7}$	$2.156954157 \times 10^{-14}$	$-2.486402486 \times 10^{-12}$
2	$6.701631702 \times 10^{-4}$	$-5.332634033 \times 10^{-11}$	$-2.871794872 \times 10^{-8}$
3	$-2.992579643 \times 10^{-1}$	$7.477905983 \times 10^{-8}$	$9.629059829 \times 10^{-5}$
4	$1.042503030 \times 10^{3}$	$2.527878788 \times 10^{-7}$	$2.060606061 \times 10^{-5}$

这些多项式曲线拟和系数中的数据都是在100≤T≤600K范围内得到的，要求输入参数的T值也在此范围内。

149
~
150

第4章 编制和调试MATLAB程序

前几章介绍了MATLAB的基本组成部分，接下来的几章将介绍各种计算问题的求解方法。本章介绍应用于数值计算中的两个附加部分：数值求解问题的编制（organizing）技术和代码纠错技术。

图4-1总结了本章的内容。前两节包含编制（或结构化）MATLAB程序解决实际的数值问题的一些建议，还涉及单个m文件中的代码结构和把一个计算任务放在多个m文件中实现的问题。本章还将对在一连串相关的示例中出现的一个大数据集进行分析，以此来介绍编制MATLAB代码的方法。

采用结构良好（well-organized）的代码有助于减少代码的错误。而且，使用良好的代码结构和采用防错性策略可以隔离代码模块中的错误，使一个模块中的错误不至于扩散到其他模块。当然，错误是很难避免的。程序中的错误统称为bug，找错和纠错的过程称为调试。本章的后半部分给出了调试（debug）MATLAB代码的建议。

读者按顺序阅读本书各章节的内容应该能够很快地学会代码的编写。本章在第三部分中将会处理更大的工程项目，体现出调试的重要性，有关调试的信息占了一定的篇幅。关于本章内容的学习，目前可以仅做浏览，当需要时再回头参考可能会更有益一些。

151

本章主题

1. m文件的组织和编排

提供了单个m文件的标准结构，讨论了m文件的一般编排方法，介绍了NMM工具箱中函数序言的使用

2. 数值问题的求解

介绍了采用逐步求精方法设计数值分析代码，并应用于为解决一个中等复杂程度项目的一连串相关的示例中

3. 调试

讨论了提高调试代码效率的策略，介绍了一些MATLAB的基本调试工具

图4-1 第4章的主题

4.1 m文件的组织和编排

根据作者的经验，编写程序并使之正常工作的时间远远超过运行程序的时间。往往大部分时间都花在调试上。而且，结构良好的代码有利于程序的调试，所以把代码的组织和调试一起介绍是很自然的。良好的组织和调试技术既可以用在简单的数值分析项目中，也可以用在复杂的数值分析项目中。建议读者不要急于编写并运行程序，最好首先设计代码，然后编写代码，在应用到重要的项目之前，最好测试程序的可靠性。很可能，后一种更系统的编程方法能够节约不少时间。

仔细地设计代码需要时间，有些人认为设计代码会延误编写代码的时间，并延长总的程序设计时间。事实上，良好的代码设计会缩短编写和调试代码的时间，而且会使所编写的程序更接近最初的目标，达到产生正确结果的目的。有几种代码设计技术，4.2.1节列出的逐步

求精（*stepwise refinement*）的方法就是其中之一。注意，代码设计是一种技术方法，而不是一种原则，不要教条化地使用。学会何时与如何使用它来求解数值问题将会很有帮助。同时，要考虑能否使用其他技术来解决同一问题。

152

本书提供的数值方法实例中绝大多数都没有涉及代码设计阶段的内容，前面提供的关于代码设计的忠告看似有些空洞。省略代码设计部分的原因是：对每个算法都进行系统设计是不现实的，这将导致本书内容繁多，以致于读者为了了解某个方法的最终实现要阅读好几页的辅助材料。

省略数值方法代码设计内容的另一个原因是因为这些方法都通俗易懂。这些方法已成功地应用在许多程序中，并且许多数学家都已经研究了这些方法的作用。所以，几乎可以预见到这些方法应用于相应问题中所产生的结果，因为这些方法就是在解决这些问题时出现的。本章介绍的代码组织在数值方法应用中将是很有用的。

4.1.1 一致性设计风格的使用

许多编程的人都会形成自己的编程风格，如：用几个连字符引出注释语句，if...end结构的执行语句采用缩进两个空格、三个空格或一个制表符的书写格式。这些习惯性的行为形成了编程风格，这类风格可使程序的视觉效果更好。读者不应忽视编程风格的重要性，使用规则的模式编码有助于编写正确的代码，而且编写的代码易于阅读和理解。结构严密、直观的代码更易于调试。

图4-2列出了本书使用的典型的m文件函数的结构。在4.1.4节中将会介绍，m文件函数一定包含序言部分。接着序言部分的第一个可执行语句用来处理所有的可选输入参数（参照3.6.1节）并检查输入变量的值域，这可视为防错性策略的部分内容（参照4.3.1节）。函数的中间部分是执行主要的计算任务。在函数的最后部分，给可选输出参数赋值。

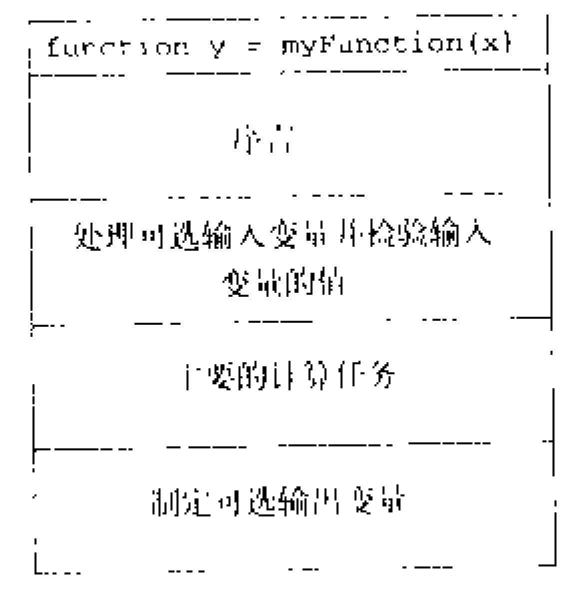


图4-2 m文件函数的基本结构

153

图4-2中的结构简明易懂，m文件函数是按顺序执行的。在4.2节中将讨论如何把代码编制成单独的模块。

4.1.2 直观的程序块和空白符

直观一致性是编程风格的重要部分。在整个程序中，类似的结构不论使用在程序的什么地方都应该呈现出相同的结构。特别是程序块结构更要求如此，比如：for循环、while循环和if...end结构都要求采用缩进的书写格式。

下例中，左边代码使用了缩进格式，右边代码不用缩进格式，左右代码的语句集相同。在本例中，大家要注意的是代码的直观结构，计算过程本身并不重要。

```
n = length(x)
if mean(x)<0
  y(1) = 0;  y(n) = 0;
  for k=2:n-1
     y(k) = x(k+1)-x(k-1);
  end
else
```

```
n = length(x)
if mean(x)<0
y(1) = 0;  y(n) = 0;
for k=2:n-1
y(k) = x(k+1)-x(k-1);
end
else
```

```
y(1) = x(1);  y(n) = x(n);
for k=2:n-1
   y(k) = 0.5*(x(k+1)+x(k-1));
end
end
```

```
y(1) = x(1);  y(n) = x(n);
for k=2:n-1
y(k) = 0.5*(x(k+1)+x(k-1));
end
end
```

左边的代码比右边的直观、更易理解。人的大脑会根据缩进编排给出的直观效果进行逻辑联想，这有助于对代码结构的理解。而且，嵌套的缩进编排和程序代码的逻辑层次是一致的，所以这对编程是十分有利的。

当然，对于编写拙劣的代码，缩进编排并不能产生任何改善质量的效果，它只能让读者对代码的逻辑结构一目了然。而且，MATLAB解释器在解释程序的时候不会受缩进的任何影响。

代码的缩进编排使用了空白符(*whitespace*)。在设计一本书的排版时，空白符就是页中没有任何字符的部分，空白符包括页边距、段间距、段的首行缩进和字间距。读者浏览本页就能看到各种空白符，空白符将每页组织成块并给读者以提示。

154

和一本书的页面的排版一样，程序中的空白符使程序的内容一目了然。代码缩进是空白符的一种应用。变量间或括号的嵌套形式中也能够使用空白符。这里列出了一条简单的MATLAB语句的4种书写形式，如：

```
y(k)=0.5*(x(k+1)+x(k-1));
                                          % 或
y(k) = 0.5*(x(k+1)+x(k-1));
                                          % 或
y(k) = 0.5*( x(k+1) + x(k-1) );
                                          % 或
y(k) = 0.5 * ( x(k+1) + x(k-1) );
```

具体使用时，空白符的个数由用户自己决定，但是空白符并不是多多益善，太多反而容易分散读者的注意力。特别地，太多的空白符会导致编辑窗口的一行容不下一条语句，而使这条语句看起来不紧凑。最重要的是用户在编写代码的时候，把空白符用在该用的地方。

本书程序清单列出的函数如果增加空白符会更加明了。这可能会使一个程序清单要占据多页。为了节省纸张并让读者从整体上掌握m文件的内容，且避免读者不断地翻页[⊖]，本书并没有过多地使用空白符，而是在必要的时候才用。

4.1.3 有意义的变量名

有意义的变量名易于代码的阅读，特别地，当代码编完后发现错误时，它使程序的调试更加简单。MATLAB对变量名的选取有很大的灵活性，既然使用有意义的变量名有助于编制代码，读者就不要局限于使用普通的变量名，如x、y等。但是，也不要使用太长的变量名，这样会使程序显得很繁琐。

给定管道的内直径和厚度，计算管道的内径和外径。下面是实现这一计算的两种方法，它们分别采用了不同的变量名。先用手盖住右边的代码，仔细研究左边的代码后再看右边的代码。

155

⊖ 参考文献[47]中Kernighan和Plauger建议把程序代码限制在一页范围内，把那些很长的程序模块划分成多个子模块分别处理一部分作业，这条建议值得采用，当程序模块太长时，有必要把它分成多个子模块。

```
d = 5;                          d_in = 5;
t = 0.02;                       thick = 0.02;
r = d/2;                        r_in = d_in/2;
r2 = r + t;                     r_out = r_in + thick;
```

尽管两组代码都正确，但是右边的代码表达得更明确。左边的代码在变量的名字上没有明确地表示出d是内直径和r是内半径，这种容易造成混淆的变量名可能会引发计算中的错误。

4.1.4 文档资料

只要程序不是马上要用且用完就扔掉，那么，文档资料就是十分关键的。编写文档资料是指编写代码过程中的注释语句，包括描述函数输入输出参数和程序执行步骤的注释语句。要认识到编写文档是编写程序代码的不可分割的一部分。特别要注意，文档资料要在编写代码的同时进行，而不要在编写完代码且在使用程序的时候才编写文档。在重用没有文档的程序中的代码时，还要重新找出代码所实现的逻辑和数学功能。编写得好的文档资料不仅易于理解和预防错误，而且有助于扩充代码功能。

有两种形式的文档：外部文档和内部文档。外部文档由描述程序的原理和所执行操作的各种报告组成；内部文档由程序中语句的描述和输入输出变量的描述等部分组成。内部文档也就是代码后的注释语句，它与代码共存。

注释语句 MATLAB中的注释语句由%开始，一直到本行行末。执行程序时，解释器不解释注释语句。

注释语句前面可以空出一行，以此来标识一段程序代码的开始。程序清单4-4中的函数H2Odensity的第25、34、39行就是这种注释语句。比较短的注释语句和MATLAB语句处于同一行，如程序清单3-13中函数fsum的最后3行。

要在编写代码的同时写注释语句，不要在编完代码并运行后才去添加注释语句，只有这样才能使整个文档的形成和编写代码成为一个同步的过程。即使不能保证注释语句完全反映出代码的功能，但这样做也能增加注释语句对程序功能的反映程度。另外，注释语句能够反映代码的输出结果，而且调试时若代码没有完成注释语句所说的功能，就可以发现代码中的错误。 156

Kernighan和Plauger在文献[47]中提出了编写文档资料的较好建议。下面是对他们所提建议的总结：

1. 当程序不正确时，文档资料写得再好都没有任何价值。
2. 与代码不对应的文档资料没有任何价值。
3. 所以，任何代码要有自己的文档资料。若没有，重新编写代码并同时写文档，而不要在原有代码的基础上添加文档资料。优秀的代码所需注释语句比差的代码要少。
4. 注释语句不只是对代码的机械表达，它应包含代码不能直观说明的意义。
5. 有助于记忆的变量名、标签和逻辑结构清晰的布局自身就有文档的作用。

函数的序言 函数顶部的注释语句称为函数的序言。在大部分简单的函数中，序言所占的行数比实际执行计算的MATLAB语句的行数还要多一些。这不是问题，为函数功能提供详细文档的目的是为了增强函数的可用性（usability）。我们知道注释语句所占的这些额外磁盘空间是没有意义的，所以，使用注释语句要适量，不能使用过多的注释语句！

编写序言有助于使用者阅读m文件源代码，它支持help和lookfor命令。当用户在命令

提示符后输入:

```
>> help funName
```

时，调用funName的序言。此时，MATLAB系统会打印出*funName.m*文件前面的注释语句，准确地说，是打印出函数定义和第一个空白行之间的注释语句。为了支持在线帮助，函数定义之后或序言内不能紧接空白行。

本书中所有具有实际用处的函数都有序言，序言包括摘要、对照表和输入、输出参数描述四部分。只有那些仅有一两行语句的简单的公式计算程序才使用一行摘要性的注释语句作为序言。

图4-3中给函数H2Odensity的序言（参照程序清单4-14）添加了注释。下面介绍序言中使用到的各种主要元素。读者最好能把下列的描述与图4-3结合起来看。

157

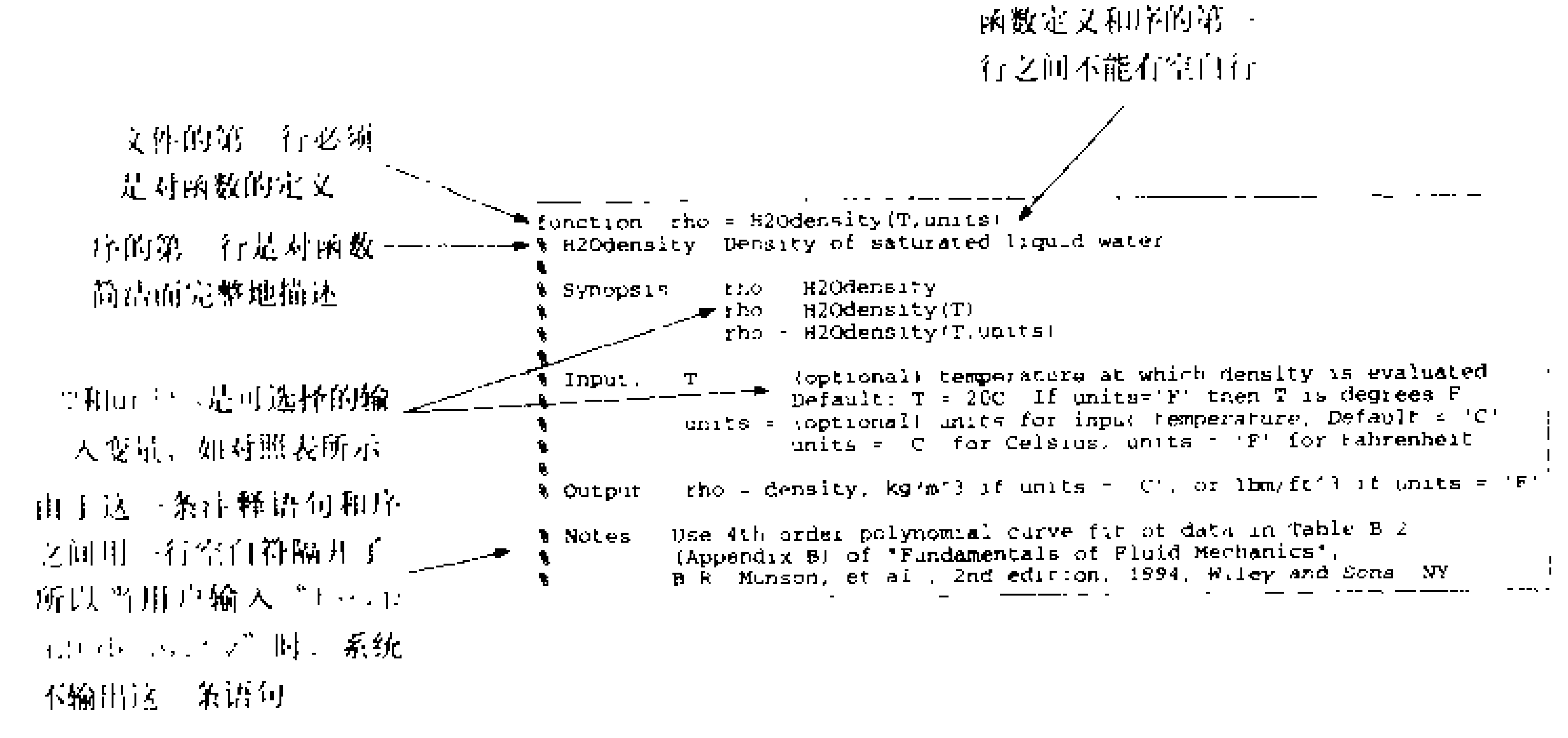


图4-3 本书中典型的m文件序的布局

摘要部分：摘要部分就是紧接函数定义的那一行注释，其内容包括这段程序的名称和函数的功能描述。紧接着的几行注释语句是摘要部分的附加说明，这些语句用来描述函数附加的功能。函数的第一行注释语句对支持MATLAB内置的lookfor命令来说特别重要。

当用户输入“lookfor *findString*”时，MATLAB系统在所有的m文件的第一行注释中搜索*findString*字符串。一旦找到匹配的字符串时，在命令窗口中打印出包含对应字符串的那一行注释语句。所以，在编写摘要行的时候需要特别仔细，语言要精练，以便使用lookfor命令能够找到对应函数的信息。摘要是包含用户所需关键字的描述性文字。同理，m文件的名字也很重要，m文件的名字也就是相应函数的名称。若不在序言的第一行说明语句中写出函数名，则用户用lookfor函数将查找不到相应的m文件而且在使用在线帮助时将很难找到所需的详细信息。参照lookfor函数可获得更多信息。

对照表部分：对照表是对函数各种调用方式的列表。若没有可选的输入输出变量，那么对照表部分只有一行注释语句。

输入部分：这部分描述每个输入变量，标记可选输入变量并指出这些变量的默认值。

输出部分：这部分描述每个输出变量。若没有返回变量，则对屏幕输出的打印信息或图形窗口中的绘图信息进行简要描述。

158

要使序言能够对使用者有用，就要求序言能准确地描述函数如何工作并介绍如何使用函数。当输入、输出变量或函数所使用的算法改变时，注释语句也需要更新。

4.2 编制数值解法程序

有些数值计算问题只要进行简单规划并通过交互方式就可以在MATLAB中实现。若用户对MATLAB很熟悉并对所求问题很了解的话，就能够通过敲入少量的语句而得到所求的结果。对这类简单问题的求解，用户可以将问题分析、初步设计和详细设计综合在一起进行。当待求解问题比较复杂时，由于分析难度的加大或处理步骤极多，在编写代码前最好先进行代码设计。

这里提出的方法包括4个基本步骤。首先，在计算机上实现解法前先作好计划。建议读者使用一种简单的*逐步求精*（*stepwise refinement*）的策略，确定主要的步骤并把它们逐步细分成更小的有明显（或接近得到明显解）解的步骤。第二，根据逐步求精所得结果，按任务定义单独的程序模块，每个模块完成一个小任务或一组（只几个）关联的小任务。这两步属于数值问题求解的设计阶段的内容。第三，实现每个程序模块，即编制MATLAB代码。若设计任务完成得好，编写MATLAB代码将会很简单。第四，测试代码。数值方法在应用到实际问题前，需要先测试。

下面是一个中等难度的例子，接下来是这个例子的解法。

例4.1　格伦峡谷大坝（Glen Canyon Dam）数据的分析

美国科罗拉多河（Colorado River）的格伦峡谷大坝的管理目标是实现最大化电力供应能力。通过调节流过发电涡轮机的水流速（flow rate）来匹配用电量需求，用电量需求在每一个小时之内也是动态变化的。大坝下游的峡谷和河滩受水流速的严重影响，为了防止河水对下游产生灾害，需要监视流过大坝的水流速并及时报告。

[159]

测试数据是一年内大坝每小时的水流速。通过画出每小时的水流速图，计算并画出每天乃至每周的水流速。给定每小时测定的水流速，$Q(t)$，一年中的第i天的水流速为

$$Q_{d,i}=\frac{1}{24}\int_{t_i}^{t_i+24}Q(t)\,\mathrm{d}t \tag{4-1}$$

这里t_i的单位是小时。同样，第j周的平均水流速为

$$Q_{w,j}=\frac{1}{168}\int_{t_j}^{t_j+168}Q(t)\,\mathrm{d}t \tag{4-2}$$

这里$168=7\times24$是每周的小时数。

该任务在概念上并不难，但要完成它需要好几个步骤，并且要求积分$Q_{d,i}$和$Q_{w,j}$。本章后面的例题将按顺序分别讲解这个例题的设计、实现和测试。

4.2.1 逐步求精

给定一个计算任务的抽象描述，如上述例题，通常需要把它分解成几个连续的小任务。*逐步求精*就是指把大型的任务分解成连续的几个小任务，再把这些小任务分解成更简单且能直接得到结果的更小任务的一种软件设计策略。这种方法也叫*自顶向下*（*top-down*）设计或*分治法*（*divide and conquer*）。

并非所有的项目都适合采用逐步求精方法来求解。对于那些目的不明确的项目，要采用

更复杂的工具，只有对那些目的明确的特定任务才能使用逐步求精的方法。

思考下列作业：画出从0到2π的正弦波形。这个简单的作业按如下3个步骤顺序执行：

1. 产生向量x，向量x中的元素是从0到2π
2. 计算$y_i = \sin(x_i)$
3. 画出y_i-x_i图

把一个计算问题分解为三个子任务，这是使用初级逐步求精方法的一个应用例子。这三个子任务都能直接用MATLAB语句实现，所以不需要继续分解。下表[⊖]列出了上面三步用

160

MATLAB语句的实现过程。

任务	→	MATLAB语句
1.产生元素在0到2π之间的向量x		`x = linspace(0, 2*pi);`
2.计算$y_i = \sin(x_i)$		`y = sin(x);`
3.画出y_i-x_i图		`plot (x, y);`

处理更复杂的任务时需要使用多级逐步求精方法。在编写代码前注意要自己动手对任务进行逐步求精。在正式开始编写和调试程序之前，对求解任务进行概念上的分析有利于用户掌握整个任务的逻辑结构。

例4.2 用逐步求精的方法分析格伦峡谷大坝的水流数据

例4.1描述了格伦峡谷大坝的水流数据分析。本例中，使用逐步求精的方法把分析的目标进行分解。首先，给出问题的最综合的陈述：

画出水流数据图。

这句话是下一级别逐步求精的基础。通过回答“画水流数据图需要什么？”可以得到一个更详细的任务列表。在求精过程中每个级别的求精阶段只增加明显的步骤。为了画图，需要下列步骤

- 从文件中读出水流数据
- 画出每小时水流速数据-时间图
- 画出每天水流速数据-时间图
- 画出每周水流速数据-时间图

先根据每小时水流速数据计算每天和每周的水流速数据，再画出对应图形。为了使任务更明显，把上面的步骤重新组织得出下列4个主要任务：

I. 把数据从文件读入MATLAB的变量中
II. 计算每天的平均水流速
III. 计算每周的平均水流速
IV. 画出每小时、每天、每周的水流速图

上面每个步骤仍然可以细化。从文件读入数据（任务I）包含：

a. 将数据装入MATLAB的矩阵变量中
b. 把矩阵中的数据分别拷入时间向量和水流速向量

161

对4个主要的任务分别细化得下列任务集：

I. 把数据从文件读入MATLAB的某个变量
  a. 将数据装入MATLAB的矩阵变量中

⊖ 语句plot(0:pi/50:2*pi, sin(0:pi/50:2*pi))或fplot('sin', [0 2*pi])能够把3个步骤用一条语句实现。

b. 把矩阵中的数据分别拷入时间向量和水流速向量

II. 计算每天的平均水流速

a. 确定数据集中的总天数

b. 对每天的时间求水流速的积分

c. 除以每天的总时间

III. 计算每周的平均水流速

a. 确定数据集中的总周数

b. 对每周的时间求水流速的积分

c. 除以每周的总时间

IV. 画出每小时、每天、每周的水流速图

在例4.3中把这些经过细化后的子任务翻译成了MATLAB语句。

4.2.2 实现：单程序多m文件

把长而复杂的数值计算分解成若干个单独的代码模块是明智的，其中，每个模块用来完成一个特定的任务。对于MATLAB程序，模块就是m文件函数。由于m文件脚本会产生副作用而且没有输入输出参数，所以它不适合用于模块化编程。

分解大型项目的一个好处就是每个模块能够独立开发和测试。而且，通用模块可以在其他项目中重用，当改进这个模块时（如提高效率），受益的不只是一个项目，而是所有那些共享这个模块的项目。

上一节介绍的逐步求精的方法就是一种把一个大型的任务分解成易于理解和实现的几个子任务的技术。列出任务的最终结果清单是把任务分解为模块的第一步。每个模块完成一个或者多个子任务，子任务细分的程度要求用户自己判断。实际上，把任务分解成模块并不是最后的结果。分解的目的不是为了产生大量的m文件，而是为了通过有效分配子任务来实现更大的目标任务。编程经验有助于对分析进行分解细化。

一旦开始编写代码，程序员对问题的认识会更深入，有时能发现逐步求精所得的分解不是最优分解，有时需要对模块重新设计或需要对任务进行再分解。模块化编程比单一模块编程更易于变更和重新安排的这一特性使它能应用于这两种情况中。 162

模块是独立的代码块，编程人员通过使用输入和输出变量来实现模块之间的通信，根据模块所要完成的任务来确定需要哪些输入值和返回哪些输出结果。在编写代码前，程序员需要在定义模块的时候指定输入和输出参数。

考虑到以上几个问题，在划分一个编程任务为若干代码模块时，建议用户采用下列步骤：

0. 使用逐步求精方法把任务分解为子任务

1. 规定一组相关联的子任务为单独的模块

2. 列出每个模块所需的输入和输出参数

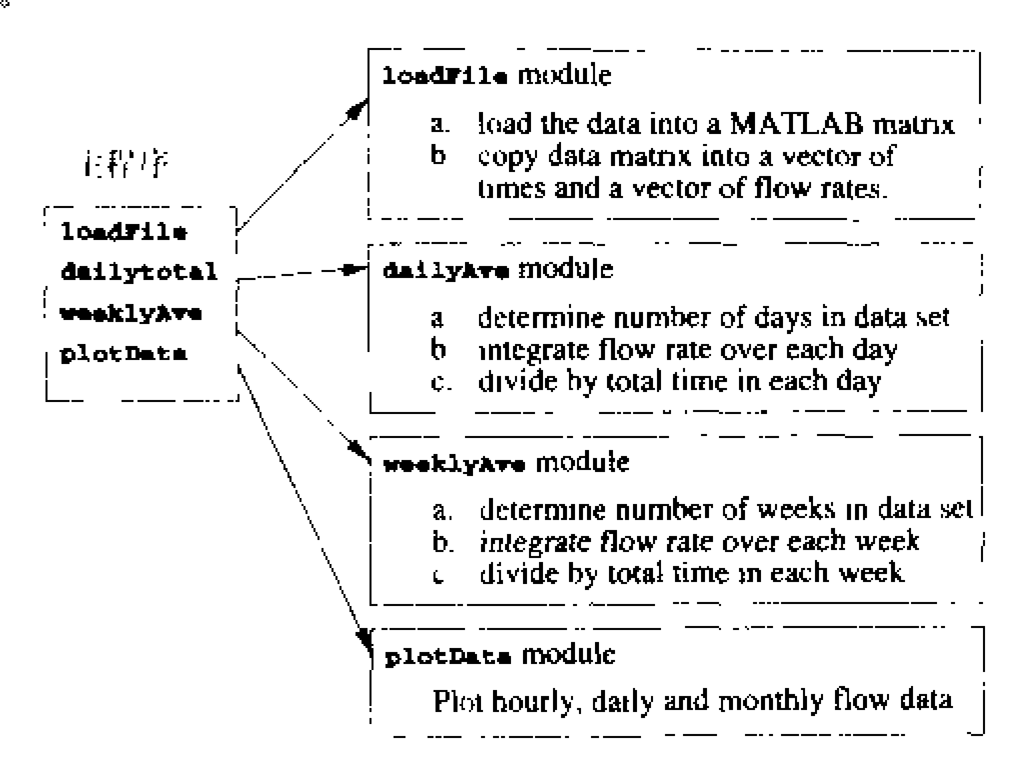
3. 看看能不能再改进，若需要的话，重复步骤1和2

对于实际的工程项目，要重复进行上述步骤，直到产生可用的程序设计结果。在设计阶段更改设计比完成代码编写后进行更改更有助于提高效率。

例4.3 实现格伦峡谷大坝数据的分析

例4.2给出了格伦峡谷大坝数据分析任务的子任务详细清单。现在把这些子任务用

MATLAB程序实现。



步骤1：规定相关任务为模块　编制代码的第一步是把模块和对应的任务关联起来。下图列出分解后对应的任务：

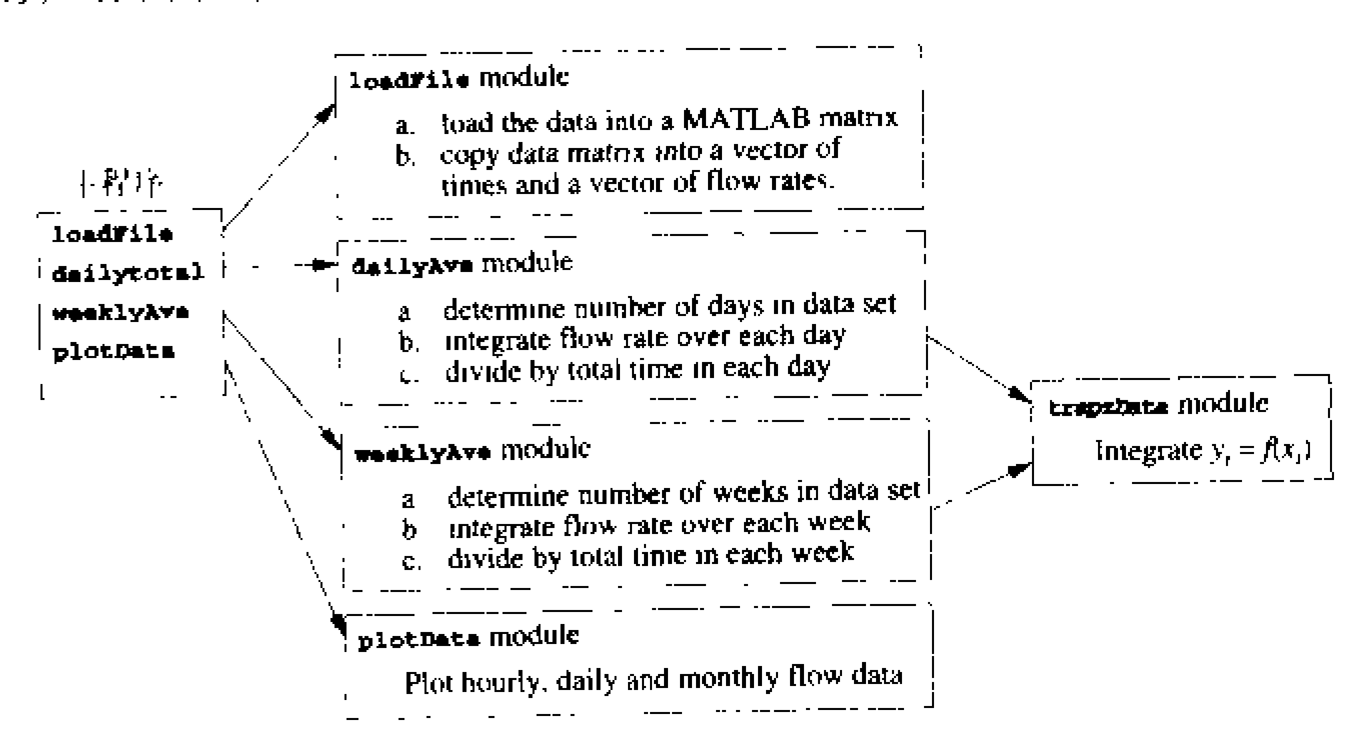
163

每个模块分别用一个方框表示，方框内包含该模块要完成的任务的列表。模块的表示方式并不重要，在实际工作中，通常用户自己会用笔在稿纸上画出上面的图例。

处于最顶层的主模块，或称驱动模块，用于定义分析的结构。如上图所示，主模块主要是由对下层模块的调用组成。LoadFile、dailyAve、weeklyAve和plotData处理所有工作。

上面的模块定义构成了初步的分析计划。现在正好处于改进代码结构的阶段。特别地，在多个独立模块中的重复子任务，提供了简化程序的机会。公用的子任务可以放在独立的共享模块中。

在本例中，模块dailyAve和weeklyAve都需要对水流量求和。程序清单11-3的函数trapzDat用来求$y = f(x)$的离散积分，它使用梯形法则，这里不作深入介绍。只要模块按规范运行，它们就能被视为（暂时的）输入输出系统（一个"黑盒"）。使用函数trapzDat计算积分任务产生下面的模块图：



现在，完成了任务分解。没有什么能明显地再简化了。下面介绍模块定义进一步的改进。

步骤2：列出每个模块的输入输出参数 为每一模块图添加输入、输出参数。为了编写文档方便，在这个级别上精练模块，输入、输出参数采用英文单词定义。当完成模块设计后，使用有效的MATLAB变量名表示输入、输出参数。下图列出了每个主模块的输入输出参数： 164

loadFile module
input: file name
output: hour vector, hourly flow vector
tasks:
a. load the data into a MATLAB matrix
b. copy data matrix into a vector of times and a vector of flow rates.

plotData module
input: hour, hourly flow, day, daily flow, week, weekly flow
output: plot of flow rates
tasks:
Plot hourly, daily and monthly flow data

dailyAve module
input: hour vector, hourly flow vector
output: day vector, daily total vector
tasks:
a. determine number of days in data set
b. daily flow = **trapzdat**(hourly flow)
c. daily ave = daily flow/24

weeklyAve module
input: hour vector, hourly flow vector
output: week vector, weekly flow vector
tasks:
a. determine number of weeks in data set
b. weekly flow = **trapzdat**(hourly flow)
c. weekly ave = weekly flow/hours per week

步骤3：检查各模块的设计并进行修订 前面描述的4个模块可以作为一个m文件的工作集。观察输入输出参数后可知，这些模块是可以改进的。在模块表示为MATLAB代码之前，这里给出了改进的一些方面，对设计的其他改进留给读者作为练习。检查时要依次考虑到每个模块。本书为了节省空间，只打印出模块所包含函数的序言，NMM工具箱中提供了函数所有的源代码。

`loadFile`模块 `loadFile`模块的最终版本和步骤2中的描述不相同。实际上，不需要编写`loadFile`模块的代码，因为内置函数`load`能够读取数据文件中的每小时水流速数据。使用内置函数`load`（不是命令）的格式从`gc87.dat`文件将数据读到包含两列的矩阵中。

```
D = load('gc87.dat');
```

后面的分析会发现把时间和水流速放在单独的向量中会很方便[⊖]，这能通过把D矩阵中的数据拷入h和f两向量中来实现：

```
h = D(:, 1);        f = D(:, 2);
```

给h和f赋值后，矩阵D在内存中就是多余的了。若内存有限，可以省去创建向量h和f这一步，而直接用`D(:, 1)`和`D(:, 2)`来代替使用两个向量。注意到`gc87.dat`中的小时都是整数，所以不必保存向量h。当需要小时向量时，通过语句（`1: length(f)`）'能够直接创建。这样做能节省内存空间，但是对于非整数形式的时间数据，使用这些代码将会出错。所以，本例中仍需保留向量h。模块`loadFile`的最终代码如下所示： 165

```
D = load(fname);              % Read data from file into D matrix
h = D(:,1);  f = D(:,2);      % Copy data into hour and flow rate vectors
D = [];                       % Free memory used by D matrix
```

最后一行释放矩阵D内存的语句不是必需的，这样做是为了防止后面遇到更大的数据文件导致内存不够。

根据到目前为止的描述，是否可以把`loadFile`模块编码为单独的m文件函数呢？这是可

⊖ 尽管没有绝对必要。

行的。由于loadFile模块的功能只用上述3条MATLAB语句就可实现，所以可直接把这3条语句放入主程序中。

dailyAve和weeklyAve模块　计算每天和每周的平均水流速的公式比较相似、只是定义的时间间隔不同(比较方程（4-1）和（4-2）)。由于这两个模块非常相似，可以把它们并入到同一个函数中，把时间间隔作为输入参数。要求读者自己来实现二者的合并（参照习题8)。这里只介绍使用单独的dailyAve和weeklyAve函数这种笨拙一点的方法。程序清单4-1列出了函数dailyAve和weeklyAve的序言。

trapzDat模块　使用梯形法则计算方程（4-1）和（4-2）中的离散数据的积分，用trapzDat函数来实现，函数的输入是向量x和$f(x)$数据、输出是$\int f(x)\,dx$。在11.2.1节中，有关于trapzDat函数的详细介绍。

由于只使用文件数据中部分的时间和水流数据来定义每天和每周的水流速，而函数trapzDat用所有的输入数据计算积分、这就要求限定trapzDat函数的输入时间范围和水流数据范围。下面列出了dailyAve中处理输入问题的方法：

```
function [d,fave] = dailyAve(t,f)
...  %  function prologue

n = length(f);     %  Number of points in input data
... % additional statements to manage data sets with uneven number of hours

for i=1:24:n
  k = k + 1;
  dayAve(k) = trapzDat(t(i:i+23),f(i:i+23));  % integrate flow*dt
end
dayAve = dayAve/24;
```

166

程序清单4-1　分析格伦峡谷大坝数据中使用的函数dailyAve和weeklyAve的序言

```
function [days,dayAve] = dailyAve(dtime,flow)
% dailyAve   Compute average daily flow from hourly flow data
%
% Synopsis:  [days,dayAve] = dailyTotal(dtime,flow)
%
% Input:     dtime = time in days from the begining of the year.
%                    Fractional values correspond to hours during the day
%            flow  = flow rate (cfs)
%
% Output:    days   = vector of integer days from beginning of the year
%            dayAve = total flow rate for each day of the year

function [week,flowAve] = weeklyAve(dtime,flow)
% weeklyAve  Compute average weekly flow from hourly flow data
%
% Synopsis:  [weektot,flowAve] = weeklyTotal(dtime,flow)
%
% Input:     dtime = time in days from the begining of the year.
%                    Fractional values correspond to hours during the day
%            flow  = flow rate (cfs)
%
% Output:    week    = vector of integer weeks from beginning of the year
%            flowAve = average flow rate for each week of the year
```

子表达式t(i:i+23)和f(i:i+23)从向量t和f中选择了24个元素序列。这些语句显示了函数trapzDat积分的范围。

plotData模块 初步的设计中把画图放在单独的plotData模块中，这就要求把所有的水流速数据从主程序传到画图模块中。由于在MATLAB中画图很简单，可以直接把画图语句合并到主程序中。即把模块plotData执行的任务添加到m文件riverReport（主模块）中。

riverReport主程序 程序清单4-2列出了水流数据分析的主函数riverReport，这个函数有一个输入变量——文件名，这个文件包含了要分析的数据。把文件名作为变量的好处就是能够把同一段代码用于分析任何一年的数据。 167

程序清单4-2 函数riverReport处理所有的数据并画出图形结果

```
function riverReport(fname)
% riverReport  Compute and plot summary of river flow data
%
% Synopsis:  riverReport(fname)
%
% Input:     fname = (string) name of file containing flow data
%
% Output:    Printed summary statistics and plots of flow data vs. time

% --- Read data into working vectors
D = load(fname);            %  Read data from file into D matrix
h = D(:,1);  f = D(:,2);    %  Copy data into hour and flow rate vectors
D = [];                     %  Free memory used by D matrix

% --- Compute daily total and weekly average flow rates
[day,flowPerDay]   = dailyAve(h,f);
[week,flowPerWeek] = weeklyAve(h,f);

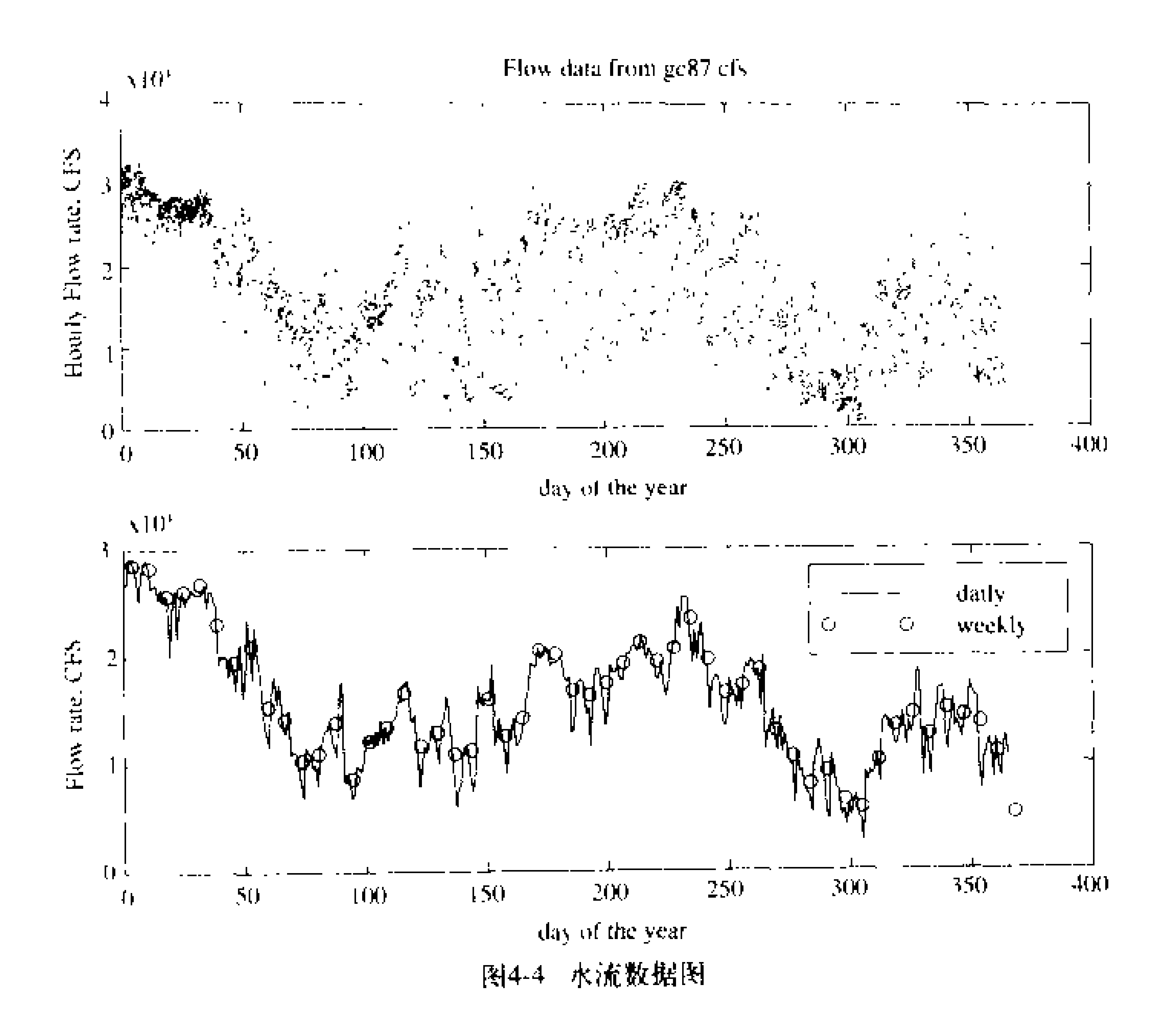
% --- Plot flow rates
subplot(2,1,1)
plot(h,f,'b.','markerSize',2);
xlabel('day of the year');  ylabel('Hourly Flow rate,  CFS');
title(sprintf('Flow data from %s',fname));

midweek = 7*week - 3.5;    %  vector of days in the middle of the week
subplot(2,1,2)
plot(day,flowPerDay,'r-',midweek,flowPerWeek,'ko');
xlabel('day of the year');  ylabel('Flow rate,  CFS');
legend('daily','weekly');
```

函数riverReport把数据文件名作为参数调用load函数，然后把由load函数返回的一个矩阵中各列的内容，分别拷入向量h（小时）和向量f（每小时水流速）中。给矩阵赋空值从而释放矩阵占据的内存空间。用函数dailyAve和weeklyAve分别计算每天和每周的平均水流速，最后，计算结果用图形表示。运行函数riverReport的典型调用语句如下所示：

```
>> riverReport('gc87.dat')
```

产生如图4-4所示的结果。画出最后的图形并不能说明函数riverReport和相关的支持函数是完全正确的。事实上，这些计算中至少存在一个缺陷（参照练习6）。



168

图4-4　水流数据图

4.2.3　测试

在把m文件应用到实际问题前必须保证它们的输出结果是正确的。若计算机器零部件应力的程序出错，那么零部件在运作中可能会受到损坏。或者，零件的超高安全标准设计会导致不必要的成本浪费。其他情况下，错误的计算可能对人员和财产造成伤害。基于上述原因，我们不能草率地给计算程序结果的正确性下结论（应该进行适当的怀疑！）。

检查计算结果正确性可以采用手工或自动的办法，而且都要进行独立测试。检验设计的正确性一个常用的方法是利用模型进行物理试验。由于通常情况下，物理试验花费较多资金，所以对那些作为设计前提的数值计算，在应用它之前先测试其正确性是很有意义的。

169

模块化编程为程序测试提供了有利的框架。单个的模块可以单独测试或联合测试。典型地，一个诊断程序调用带输入变量且结果已知的模块，若产生的结果不是期望值，那么就要仔细地检查被调用模块，或要设计更好的专门的诊断程序来测试它。在程序开发的这一阶段使用交互调试器会大有帮助。一旦发现并修改了模块中的错误，该诊断程序在以后还能够使用。若对模块进行了改进，还可以再次运行诊断程序，这样，将模块并入更复杂的分析前不会出现很明显的错误。

有时把诊断计算也包含在模块本身中，这是有意义的。一旦模块达到规范要求，通过把诊断计算程序放入if...end结构或把诊断语句改成注释语句就能够关闭诊断程序。诊断的一个最简单的形式是在迭代过程的每一步打印结果。通常当程序正常工作时，这种打印是不必要的，但当它应用到新的问题时，具有打开打印和关闭打印功能是有益处的。给输入参数

列表中加一个简单的标记就能实现打印的打开和关闭操作。例如，程序清单6-4中的函数bisect中的verbose标记和程序清单6-6中的newton函数就是这方面的示例。

编写诊断程序时，最好使用能产生精确的解析结果的测试问题（test problem）。本书通篇使用的示例(demo)函数就是测试程序一个例子。例如，程序清单6-5中的函数demoNewton，程序清单12-2中的函数demoEuler，程序清单11-2中的函数demoTrap。下面例子也是使用诊断程序的示例：

例4.4 测试trapzDat函数

函数trapzDat使用梯形规则计算一个离散数据集的积分。对连续函数$f(x)$，梯形规则如下所示：

$$I=\int_a^b f(x)\,\mathrm{d}x=\frac{h}{2}\sum_{i=2}^{n}(f_i+f_{i-1})+O(h^2)$$

其中，h是f在x轴上的积分间隔

$$h=\frac{b-a}{n-1}$$

170

n是数据集中的数据点的个数。若x_i不是均匀分布的，那么

$$I\approx\frac{1}{2}\sum_{i=2}^{n}(f_i+f_{i-1})(x_i-x_{i-1})$$

程序清单11-3中的函数trapzDat对给定的数据集（x_i，$f(x_i)$）计算上面公式的值。11.2.1节对函数trapzDat作了附加介绍。本例的目的是向读者介绍如何检验trapzDat计算的正确性。检验计算模块正确与否并不要求对模块内部作详细了解，而应基于对函数的功能描述和在代码文档中给出的输入输出参数进行测试。

函数trapzDat的一个简单测试就是计算已知函数$f(x)$的积分，它的一个测试问题就是函数trapzDat使用的梯形规则存在非零截断误差，导致使用梯形规则计算出的实际值与精确值存在误差（这个误差不是程序问题造成的）。解决办法是把分段常数函数(piecewise-constant function)与梯形规则结合使用，它的截断误差为零[⊖]。例如，下列程序使用函数trapzDat生成$f(x)=1$的准确结果：

```
>> x = linspace(0,1);
>> f = ones(size(x));  %  constant f(x)
>> s = trapzDat(x,f)
s =
     1
```

函数trapzDat的序言说明它的输入数据可为任意间隔。前面的测试使用了均匀间隔的数据。程序清单4-3中的trapzDatTest函数使用3种形式测试函数trapzDat是否能够准确地计算对常量函数（constant function）在x非均匀间隔上的积分。第一个测试是对偶数个数据计算$f(x)=\pi$的积分。第二个测试是对奇数个数据计算$f(x)=\pi$的积分。第三个测试是计算0到1区间中x取随机间隔值时$\int_0^1 \pi\mathrm{d}x$的积分。测试结果在后面给出。

171

⊖ 梯形规则生成次数小于等于1的分段多项式$f(x)$的精确结果。分段常量由读者视方便而选取。

程序清单4-3 函数trapzDatTest用来测试函数trapzDat的正确性

```
function trapzDatTest
% trapzDatTest  Verify trapzDat function for different types of input
%
% Synopsis:  trapzDatTest
%
% Input:     none
%
% Output:    Print out of test results

% --- Integral of a constant
fprintf('Integrate a constant function\n');
x = linspace(-5,7,8);        %  Test 1:  even number of intervals
f = pi*ones(size(x));        %  constant function
s = trapzDat(x,f);
err = s - pi*(max(x)-min(x));
fprintf('\tEven interval test:    s = %8.4f   error = %10.3e\n',s,err);

x = linspace(-5,7,9);        %  Test 2:  odd number of intervals
f = pi*ones(size(x));        %  constant function
s = trapzDat(x,f);
err = s - pi*(max(x)-min(x));
fprintf('\tOdd interval test:     s = %8.4f   error = %10.3e\n',s,err);

x = sort(rand(5,1)-0.2);   %  Test 3:  randomly spaced x data
f = pi*ones(size(x));
s = trapzDat(x,f);
err = s - pi*(max(x)-min(x));
fprintf('\tRandom spacing test:   s = %8.4f   error = %10.3e\n',s,err);

% --- Truncation error test
fprintf('\nTruncation error test\n');
fprintf('   n       integral            error         alpha\n');
xmin = -5*pi/7;  xmax =  3*pi/5;  % oddball starting and stoping points
hold = -5;       errold = 0;      % initialize test for first try

for n = [10  17  25  100  315  2001  4522]
  x = linspace(xmin,xmax,n);
  f = sin(x);
  s = trapzDat(x,f);
  err = s - (cos(x(1)) - cos(x(n)));
  fprintf('%5d  %15.12f  %12.3e',n,s,err);
  h = (xmax-xmin)/(n-1);
  if hold<0
    fprintf('\n');            %  first iteration, skip test
  else
    fprintf('  %9.5f\n',log(err/errold)/log(h/hold));
  end
  hold = h;  errold = err;    %  prepare for next iteration
end
```

172

另一个测试是用来判断函数trapzDat的截断误差是否和h有关，理论分析的结果表明截断误差是与h相关的。5.3.2节的开头部分和例11.6中介绍了这个测试的基本原理。对不同的h值，由积分产生的误差E得到：

$$\alpha = \frac{\log\left(\dfrac{E(x,h_2)}{E(x,h_1)}\right)}{\log\left(\dfrac{h_2}{h_1}\right)}$$

这里α= 2时，使用梯形规则。trapzDatTest函数用梯形规则计算$\int_a^b \sin x dx$在x从a = $-5\pi/7$到b = $3\pi/5$、并随h值渐减的积分。这样选择积分区间是为了包含x的负数值并避免获得太简单的积分结果。对每个h值，积分的数值计算结果与精确值相减得到绝对截断误差。对连续的h值计算α值。运行trapzDatTest函数得到：

```
>> trapzDatTest
Integrate a constant function
Even interval test:   s =  37.6991  error =  0.000e+00
Odd interval test:    s =  37.6991  error = -7.105e-15
Random spacing test:  s =   1.6643  error =  0.000e+00

Truncation error test
   n       integral           error        alpha
  10   -0.308937718819     5.535e-03
  17   -0.312725683473     1.747e-03     2.00419
  25   -0.313696787292     7.760e-04     2.00152
 100   -0.314427222335     4.559e-05     2.00033
 315   -0.314468276181     4.531e-06     2.00002
2001   -0.314472695792     1.117e-07     2.00000
4522   -0.314472785626     2.186e-08     2.00000
```

前面3个测试结果表明使用函数trapzDat获得了常量函数积分的精确值（在舍入范围内）。最后的测试结果表明alpha值随着点数n的增加越来越接近2，这是我们所期望的。这4个测试结果表明函数trapzDat正确地实现了梯形规则。

4.3 调试

查错与纠错是编程中不可缺少的部分。错误是由许多因素造成的，包括编程错误和使用不恰当的数值方法。刚编写完且没有调试过的程序很可能含有导致致命错误（*fatal error*）的bug，可能导致MATLAB停止运行。思考下列multiply函数：

```
function z = multiply(x,y)
% multiply  Compute product of two arguments
z = x*y;
```

这个函数没有错误，下列语句结果正确：

```
>> multiply(2,3)
ans =
     6
```

而下列语句产生了一个重大错误：

```
>> x = 1:4;   y = 1:4;
>> multiply(x,y)
??? Error using ==> *
Inner matrix dimensions must agree.
Error in ==> MATLAB:toolbox:nmm:orgbug:multiply.m
On line 3  ==> z = x*y;
```

第二次使用multiply时，它的输入变量都是列向量，两个列向量不能直接相乘（参照7.1.1节）。MATLAB解释器无法预先知道向量x与y维数是否匹配。此时，这类错误是由输入数据产生的。这不是函数multiply本身的错误，而是函数使用上的错误。

必须把所有的致命错误都改正过来整个程序才能运行。程序能够运行后，可能仍然存在错误。找出计算中的小错误更加困难，比如只造成计算值出错的一类错误。像前面一节说的，测试的目的是为了找出错误。本节的目的是介绍找出错误和纠正错误的技术。

4.3.1 防错性程序设计

防错性程序设计（*defensive programming*）是应用各种策略保证代码正确执行。检查输入数据是否在限定范围内就是一个简单的例子。如：计算$\log(x)$的函数中对输入变量x进行非负数检查是有意义的。使用防错性程序能够识别这类计算错误。

防错性程序设计技术的一些例子如下：

- 假定输入数据可能会不正确，测试输入数据的正确性。
- 防止不可能发生的条件出现而导致计算失败。
- 对if...elseif...else这一结构序列提供匹配条件或设定默认的匹配条件。
- 提供错误诊断信息以使用户知道为什么计算会产生错误。

例4.5 防错性程序设计

程序清单4-4中的函数H2Odensity使用了防错性程序设计原理。下表描述了函数H2Odensity的特定行：

174

程序清单4-4　使用了防错性程序设计原理的函数H2Odensity

```
1   function  rho = H2Odensity(T,units)
2   % H2Odensity  Density of saturated liquid water
3   %
4   % Synopsis:   rho = H2Odensity
5   %             rho = H2Odensity(T)
6   %             rho = H2Odensity(T,units)
7   %
8   % Input:  T  = (optional) temperature at which density is evaluated
9   %              Default: T = 20C. If units='F', then T is degrees F
10  %      units = (optional) units for input temperature, Default = 'C'
11  %              units = 'C' for Celsius, units = 'F' for Fahrenheit
12  %
13  % Output:  rho = density, kg/m^3 if units='C'; lbm/ft^3 if units='F'
14
15  % Notes:  Use 4th order polynomial curve fit of data in Table B.2
16  %         (Appendix B) of "Fundamentals of Fluid Mechanics",
17  %         B. R. Munson, et al., 2nd edition, 1994, Wiley, NY
18
19  if nargin<1
20    rho = 998.2;  return;    %  Density at 20 C
21  elseif nargin==1
22    units='C';               %  Default units are C
23  end
24
25  % --- Convert to degrees C if necessary
26  if upper(units)=='F'
```

```
27    Tin = (T-32)*5/9;       %  Convert F to C; don't change input value
28  elseif upper(units) == 'C'
29    Tin = T;
30  else
31    error(sprintf('units = ''%s'' not allowed in H2Odensity',units));
32  end
33
34  % --- Make sure temperature is within range of curve fit
35  if Tin<0 | Tin>100
36    error(sprintf('T = %f (C) is out of range for curve fit',Tin));
37  end
38
39  % --- Curve fit coefficients
40  c = [ 1.543908249780381441e-05  -5.878005395030049852e-03 ...
41        1.788447211945859774e-02   1.000009926781338436e+03];
42
43  rho = polyval(c,Tin);      % Evaluate polynomial curve fit
44  if upper(units)=='F'
45    rho = rho*6.243e-2;      % Convert kg/m^3 to lbm/ft^3
46  end
```

[175]

注:

行号	代码描述
2–13	序言详细描述了输入、输出变量和函数调用的语法
26–32	检查units的if...结构提供了默认的else...错误陷阱。若第28行的elseif用else替换，则输入量是非华氏度时计算的单位都当成摄氏度。这一错误陷阱让用户明白这里使用的单位制而不要错误地使用其他单位制（如：'R'是兰金温度（Rankine）、'K'是热力学温度）。
26、28、44	变量units转换成大写字母。这允许用户输入小写"f"、来表示"F"
27、29	在从摄氏温标转换成华氏温标时，为防止覆盖变量T而创建新的变量Tin。
35–37	检查变量Tin是否在曲线拟合的范围内。若不在，给出错误信息并终止程序的执行。

4.3.2 调试工具

MATLAB 5包含了交互调试器，这种交互调试器对程序员调试程序有很大的用处。由于调试器在不同的计算机平台上运行稍微有点不同，所以这里不对它作介绍。读者有必要仔细学习参考文献*Using* MATLAB[73]中的交互调试指南。使用几个MATLAB命令也能够方便地调试程序。这里只介绍四种简单的调试命令。

type和dbtype命令 每当MATLAB遇到重大错误时，它会给出出错程序名称和出错信息。若出错函数被其他函数调用，MATLAB会给出调用函数的函数名和调用所在行的编号，这样不断地追溯出错的原因直到找到出错的语句。

一旦收到出错信息，第一步就是识别出错的位置，最基本的方法就是查看导致错误的m文件中的语句。在命令窗口输入下列语句能够查看m文件的内容:

```
type filename
```

这里*filename*是当前路径下的m文件的文件名（不带扩展名.m）。执行type命令会把整个m文件的内容显示出来。通常情况下，使用dbtype命令只显示出错行邻近的几行代码会更方便一些。dbtype有如下两种形式:

[176]

```
dbtype filename
```

它打印出整个文件的内容并附带打印行号。

```
dbtype filename start:stop
```

它打印出行号从*start*到*stop*之间每行的内容。

error函数 error函数（不要与erf函数弄混淆）把基本的错误处理加入到一个m文件函数中。它的句法如下：

```
error(message)
```

这里*message*是执行error函数时需要在屏幕上打印的字符串。error函数应并到一个m文件中，它不能在命令行中执行。函数error典型的使用方法如下：

```
if error condition test
 error(message)
end
```

这里*error condition test*是用于监测错误的逻辑表达式。执行error函数时，MATLAB列出调用error的函数名并打印出*message*，而且打印出所有调用包含这个错误的函数的函数名。这时，MATLAB终止程序的执行并把控制权返回给命令窗口。

noneg函数使用了error函数，程序如下所示：

```
function y = noneg(x)
% noneg  Returns x if x>0, otherwise prints an error message and stops
if x<0
  error('x is negative');
end
y = x;
```

当运行函数noneg时输入负数值，出现下列提示：

```
>> y = noneg(3-5)
??? Error using ==> noneg
x is negative
```

给出错信息添加相关的数字值会使出错信息含义更加明显。这要求给error函数建立一个文本串作为输入变量。下面列出一些表达式来替换noneg函数中对应的语句：

177

```
message = ['x = ',num2str(x),' cannot be negative'];
error(message);
或                                                   % or
error(['x = ',num2str(x),' cannot be negative']);
或                                                   % or
error(sprintf('x = %f cannot be negative',x));
```

采用上面第三种方法替换函数noneg中对应的语句得到如下语句：

```
function y = noneg2(x)
% noneg2  Returns x if x>0, otherwise prints an error message and stops.
%         The error message contains the value of x.
if x<0
  error(sprintf('x = %f cannot be negative',x));
end
y = x;
```

pause命令 pause命令把正在执行的m文件暂时挂起，常用于在不中断分析的情况下打印警告信息。当执行到pause语句时，MATLAB暂停程序的运行并把指针设成“P”表明程序暂停执行。这时，用户使用Ctrl-C命令终止分析或使用回车键继续执行程序。命令pause(n)使MATLAB把当前的任务挂起*n*秒，而不用等待用户的响应。

在下面的nonegp函数中，当输入值为负数时，它使用pause命令而不是终止程序的运行：

```
function y = nonegp(x)
% nonegp  Returns x if x>0, otherwise prints a message and pauses
if x<0
  disp('x is negative, press return to proceed');
  pause;
end
y = x;
```

作为例子调用nonegp（-1），看看pause命令的效果。

keyboard命令　keyboard命令挂起执行的m文件并把控制权转给用户，它比pause命令功能更强大，因为它让用户直接访问包含keyboard命令的函数（m文件）的内部变量。

keyboard命令通常用于错误已知的m文件中。当遇到keyboard命令时，命令窗口中的提示符由>>变为K>>。这时，用户能够打印出m文件中的任何局部变量，通过这些变量值来帮助查找出错原因。通常情况下，MATLAB命令窗口不能共享这些局部变量。当用户在MATALAB提示符处输入回车键时，继续执行m文件。下一例子介绍keyboard命令的使用。

178

交互调试器可以不使用keyboard命令而中断m文件的执行。当MATLAB遇到交互调试器设定的断点时，用户能够查询函数中的局部变量。

例4.6　keyboard命令的应用

程序清单4-5中的函数quadroot使用6.7节介绍的改进规则来计算二阶多项式的两个根。假设当quadroot返回复数根时，会导致其他某个函数（这里没有列出的函数）出错。查错的方法之一就是在quadroot函数中插入keyboard命令。这时能够检查二阶多项式的系数。本例的目的是介绍keyboard命令的应用，而不只是说明quadroot函数有错误。

程序清单4-5　函数quadroot用于说明keyboard命令的应用

```
function r = quadroot(a,b,c)
% quadroot   Roots of quadratic equation and demo of keyboard command
%
% Synopsis:  r = quadroot(a,b,c)
%
% Input:     a,b,c = coefficients of  a*x^2 + b*x + c = 0
%
% Output:    r = column vector containing the real or complex roots

% See Chapter 4, Unavoidable Errors in Computing, for a discussion
% of the formula for r(1) and r(2)
d = b^2 - 4*a*c;
if d<0
  fprintf('Warning in function QUADROOT:\n');
  fprintf('\tNegative discriminant\n\tType "return" to continue\n');
  keyboard;
end
q = -0.5*( b + sign(b)*sqrt(b^2 - 4*a*c) );

r = [q/a; c/q];  % store roots in a column vector
```

179

图4-5列出了MATLAB交互环境中函数quadroot对复数根的响应。输入下列语句初始化交互环境

```
>> quadroot(1, 2, 3)
```

图4-5右边的“用户输入”信息是用户在命令窗口输入的语句。注意，恢复执行函数quadroot后复数根的计算结果是正确的。

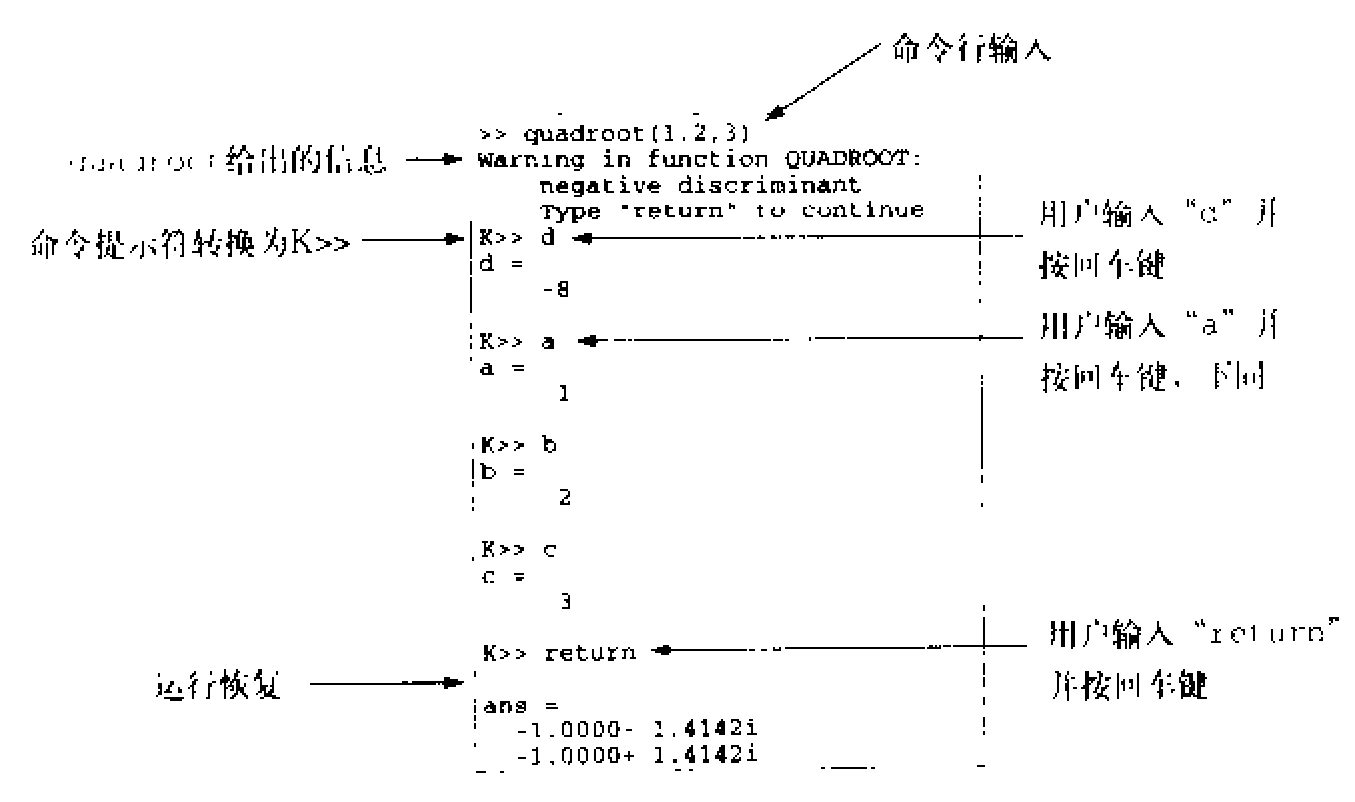


图4-5　程序清单4-5中的函数quadroot对keyboard命令的响应示例

在命令窗口输入quadroot(1, 2, 3)来初始化交互环境

4.4　小结

本章介绍了MATLAB代码的编制方法。模块化编程和详细文档资料的使用能使程序员编程达到事半功倍的效果、结构清晰的代码还有助于程序的调试，本章通过例子介绍了这些技术。表4-1列出了本章中用到的m文件。

180

表4-1　NMM工具箱中用于介绍程序组织和调试的函数

函　数	小　节	描　述
dailyAve	4.2.2	根据每小时的水流数据计算每天平均的水流速数据
H2Odensity	4.3.1	液态水的密度
multiply	4.3	计算两个变量的积
nonneg	4.3.2	若x>0，返回x的值；否则，打印错误信息并停止运行
nonneg2	4.3.2	若x>0，返回x的值；否则，打印错误信息并停止运行。错误信息中包含x的值
nonnegp	4.3.2	若x>0，返回x的值；否则，打印错误信息并暂停
quadroot	4.3.2	二次方程的根和keyboard命令的应用
riverReport	4.2.2	计算并画出河水流速图
trao2DatTest	4.2.3	以因子2递减一个数一直减到0
weeklyAve	4.2.2	根据每天的水流数据计算每周平均的水流速数据

补充读物

学习编写代码的最好的方法之一就是阅读那些有经验的程序员所编写的代码，这虽然需要时间、但确实是个很好的方法。MATLAB标准工具箱提供了大量的这种高质量的代码。它

们的文档资料比较少，但是介绍了如何把大型计算任务分解成更小的易于管理的模块。标准工具箱中的代码都很复杂，阅读这些代码的时候可以把它们的输入变量简单化，即，选择少量的输入和输出参数。

作者尝试过编写NMM工具箱中的示例m文件。细心的读者也可以尝试，因为有些m文件还有改进余地。有时，为了一页能够容纳一段代码，在一行中会塞满很多条语句；有时，简化甚至省略注释语句。所以，有必要改进代码，本书作者鼓励读者动手进行改进。编辑代码的时候，自己想想：这种改变是纯装饰性的吗？能否提高程序的可读性、可维护性和代码的可用性？建议读者首先给原始的m文件做个拷贝以便能够轻松地恢复到原始的m文件。

Kernighan和Plauger在参考文献[47]中介绍了如何编写易读且易于维护的结构化代码。他们用Fortran和PL/1语言编写了示例，这些示例能够直接转化成MATLAB代码。参考文献[46]中，Kernighan和Pike使用C、C++和Java的示例介绍了编程风格、调试策略和其他良好的软件开发习惯。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量，参照1.4节中关于分级系统的介绍。 181

1. （1+）对下列MATLAB语句，列出生成y值用到的函数。

   (a) `y = bessel(1, 3.2);`

   (b) `y = besselk(2, 5);`

提示：在命令提示符后输入“`type bessel`”可以察看函数的源代码。

2. *（2）使用逐步求精法描述计算向量x中元素的平均偏差和标准偏差的所需步骤。开发一个m文件并测试所使用的方案（解法）。要求不使用内置函数`mean`和`std`。而且，根据有限的平均偏差和标准偏差方程样本开发自己的解法。

3. （3）半导体器件由硅晶片大量生产出来，每个硅晶片包含上百个半导体器件。在硅晶片生产过程中的任意一个阶段都可能出现随机错误。晶片加工出来后，要对晶片上的每个器件进行测试，并把晶片切割成单个的器件或芯片。通过最后测试的芯片要与一个引线框相连接，然后封装成成品，这就得到了我们所看到的装配在电路板上的芯片。可用芯片与总的芯片个数之比称为成品率。

   NMM工具箱中的data目录下的`chip.dat`文件包含了一次半导体器件的生产运行数据。文件的第一列是芯片的序列号，第二列是芯片的速度限制指标，速限的单位是兆赫兹（MHz），它是指芯片能可靠运行的最高时钟速度。第二列的值等价于`NaN`时，表明该芯片的速度测试彻底失败。

   使用逐步求精的方法编写程序（可以包含多个m文件函数）计算总产量和分别满足250、300、350、400和450MHz速度规格的芯片个数。

4. （3-）同上题，打印出满足每组速度要求的芯片的序列号。

5. （3）文件`traffic.dat`包含某个城市街道上测试得到的交通流量数据。文件只有一列，用来记录交通工具通过测试点的时间（十进制的小时时间）。要求使用逐步求精的方法编写能打印出一天中每个小时经过测试点的交通流量的程序。

6. （3）例4.3中计算每周平均的水流敞据中有一个错误。由于365不是7的倍敞，所以这一年的最后一周不是7天。当一年中的周敞不为偶数时，用函数`weeklyAve`计算平均

值是错误的。图4-4中最后一周的水流数据点（图中最右边有个孤点）明显地说明了这个错误的存在。请改正这一错误。

7. (3-) 针对例4.3中的河水报告程序，编写一个新的函数dailyMinMax计算每天中的最大水流速率和最小水流速率。

8. (3) 通过消除dailyAve和weeklyAve模块中的冗余能够简化例4.3中的代码模块。水流速每天的平均值和每周的平均值的计算都是对一个确定时间间隔的平均值的计算问题。设计一个模块计算每天和每周的平均值，通过输入变量决定是对每天还是对每周求平均值。要求设计的模块包括详细地描述所执行的任务和输入、输出参数。注意，有多种方法实现这个想法。在更高层上定义模块比将dailyAve和weeklyAve组合成一个模块更简单。

182

9. (3) 同上题，实现并测试所设计模块。你的程序会不会出现练习6中提到的问题？

10. (3) 脚本indent包含了4.1.2节下面的代码及一些附加代码。使用下列语句运行indent脚本后的结果表明脚本中有一个较隐蔽的错误。

```
>> nx = 6;
>> indent
>> nx = 4;
>> indent
```

找出错误，用文字描述，并写出正确的文件indent.m。

11. (2) 内置函数interp1向一维数据集（即$y=f(x)$）中插入数据。当所插入数据点超出表中数据的范围时，函数interp1返回NaN但不会通知用户插入的数据点超出数据范围（没有任何警告信息）。如：

```
>> interp1([1 2 3 4 5],[9 8 7 6 5],11,'linear')
ans =
   NaN
```

使用函数interp1编写函数safeInterp对一维数据做线性插值，并要求实现下列防错性程序：

i) 检查interp1的返回值NaN。（当xi和yi为向量时，能否这样写？）

ii) 若发现NaN（或不止一个NaN），测试误差数据的范围（即xi>max(xtable)或xi<min(xtable)）。

iii) 若发现数据的范围出错，打印出警告信息。

iv) 根据违背范围的类型，把yi的值赋给ytable中对应的最大值max(xtable)或最小值min(xtable)。

183
~
186

第二部分 数值技术

第5章 计算中的误差

计算机系统中出现故障的原因多种多样，或者是因为硬件故障(例如磁盘驱动器故障)，或者是由于关键性数据——包括从序列号和账单总额到操作指令——的输入不正确，导致程序"可靠地"执行了错误的任务。有的错误是所运行的程序——包括从文字处理器到核电站控制系统——含有逻辑错误产生的，这会导致预想不到的结果，如数据损坏或者系统崩溃等。

软件中的错误称为bug[一]，纠正错误的操作叫做调试。软件引起的错误通常都是确定不变的，给定相同的初始数据，程序执行结果的错误也是相同的。软件越复杂，就越需要检测和纠正程序中的错误。有的时候，软件中的错误是随机出现的，因为错误可能是由计算机的状态，特别是内存中的数据引起的。

187

执行数学计算的软件和其他软件一样也会出现错误。但是，数值计算中有两种额外类型的错误：*舍入误差*（*roundoff error*）和*截断误差*（*truncation error*）。和程序中的bug不一样的是，这两种误差都无法避免。计算机计算时的舍入误差是指丢弃小数点后面的有效数字所造成的误差，因为计算机能够存储的数值数据的位数有限。截断误差出现在用离散公式近似连续的数学函数进行计算时。进行数值分析的目的就是寻找一些方法来解决某个特定的问题，使所求问题不受舍入误差的影响并且截断误差小于用户的自定义方差[二]。要设计健壮的数值算法，必须考虑舍入误差和截断误差，本书后面介绍的程序几乎都要考虑舍入误差和截断误差，本章就是为此提供背景知识。

输入数据的不确定性是影响数值计算结果的另外一类误差。例如，当输入数据来自试验测量数据或来自首次被存入内存中时进行了舍入的数据，输入数据中的误差就会带来不确定性。输入数据的这种不确定性所造成影响的重要性由数值计算的*状态*（*condition*）决定。*良态*（*well-conditioned*）的数值计算不会敏感地受输入数据的影响，*弱态*（*poorly conditioned*）或者是*病态*（*ill-conditioned*）的数值计算会使结果不可信（这还是最好的情况）或者根本就毫无价值。由于对于每种数值方法，状态的影响是不同的，所以将在后续章节对每种方法分别进行介绍。

有的读者可能会觉得奇怪，为什么在还没有介绍有用的数值方法前就去找它们的错误

[一] Grace Hopper首先使用术语bug描述一台Mark II（参照Kahaner等[43，P.35]）机器上的计算故障，Mark II是发明晶体管前使用的典型的计算机，它的逻辑电路由真空管和机电式继电器组成。有一次，Hopper发现计算出现错误的结果，她不停地追溯问题的原因，最后发现是在两个裸露的继电器之间爬行的虫子造成的。虫子被电击致死从而导致短路。Hopper把虫子粘在她的笔记本上，旁边写着，故障是由计算机中的"bug"引起的。

[二] 参考文献[76]中，Trefethen指出数值分析者错误地定义了他们在数值误差学习过程中的主要行为。本书的目的在于数值方法的实现而不仅是分析。虽然本书通篇都会涉及误差，但它不是本书的重点。

呢？这样好吗？其实，这样做很有必要。通过检查数值算法中出错的原因，为读者打好了把错误减少到最少的基础。没有数值计算的误差做基础，很难鉴别哪种算法更好。

188 图5-1总结了本章将要介绍的主题。

本章主题

1. **数值的数字表示法**

按数字格式存储数据值引入了实数的表示范围和精度问题。本节介绍实数的具体表示形式——浮点数以及数值在MATLAB中的基本存储形式。

2. **有限精度算术**

数的数字存储引起了算术运算中的**舍入误差**。舍入的位数由**机器精度**决定。本节介绍怎样管理比较操作特别是迭代计算的收敛判定准则中的舍入效应。

3. **算法的截断误差**

截断误差是指用一个相对简单的且只包含离散值的算术表达式去替换一个相对复杂的连续函数算术表达式时所引起的误差。阶符（order notation）表示控制截断误差大小的关键参数产生的影响大小。本节介绍了截断误差的基本思想，并为后面的介绍做铺垫。

图5-1 第5章的主题

例5.1 数字的自发生成

为了激励读者学习浮点计算，请思考下列MATLAB中的简单计算问题[⊖]。

```
>> format long e          % display all of the significant digits
>> 2.6 + 0.2
ans =
     2.800000000000000e+00

>> ans + 0.2
ans =
     3.000000000000000e+00

>> ans + 0.2
ans =
     3.200000000000001e+00
```

189

对精确的有理小数0.2的反复加运算产生了错误数据位数，导致结果错误。尽管错误发生在最低位上，读者会想为什么会出现错误，怎样出现的？如果反复加0.6，会出现什么情况。

```
>> 2.6 + 0.6
ans =
     3.200000000000000e+00

>>ans + 0.6
ans =
     3.800000000000000e+00

>>ans + 0.6
ans =
     4.400000000000000e+00

>>ans + 0 6
ans =
     5
```

⊖ 1993年12月6号Mark Reichelt把这个例题寄给了comp.soft-sys.matlab新闻组。

不仅没有丢失精度，而且最后结果是一个整数“5”。这是因为MATLAB中数值计算采用了有限算法。

例5.2 奔腾™ FDIV Bug

最近出现而且是最著名的数值计算错误的一个例子发生在英特尔奔腾™微处理器上，它是被一个正在研究素数的数学家Thomas Nicely发现的。他发现有些整数组合的商会产生相当大的舍入误差。他把程序在许多其他计算机上测试都没有出现问题，最后他怀疑是奔腾™微处理器硬件而不是软件的问题。

Nicely的程序说明当奔腾™微处理器执行FDIV指令处理特殊的操作数组合时会出现错误。所谓的FDIV bug是由表格的信息单元的丢失引起的，这种表格是固定不变的，建立在奔腾™处理器的片上存储器中。对于某些操作数的组合，应当精确到16位的除法，它只能精确到7位。

只有少数的操作数会导致除法出错。数值计算可能会涉及到大量的操作，谁也无法保证使用有缺陷的奔腾™芯片能够不出错。当除法出错时，原先可以预料的舍入误差会突然增大几个数量级。英特尔公司开始一直掩盖这一缺陷的重要性，但是他们最后同意替换那些有问题的芯片。读者若是想进一步知道bug的发现历史，提供的证明文件以及补救措施，请参阅文献Moler [54]。文献[20]中Edelman给出了对这个bug在数学上的分析。

[190]

5.1 数的数字表示法

由于应用上的原因，计算机只能用有限的位数来表示某个数。所以，计算机中存储的数值精度是有限的。有限精度加快了数值计算的速度，并减少了数值所需的存储空间，但是可能会出现意外的舍入（*roundoff*）——由于丢弃计算中最小有效数位带来的误差。为了解舍入误差产生的根源，我们首先看看数据在计算机中是如何存储的。

5.1.1 位、字节和字

几乎所有近代的计算机都采用二进制（即：使用基为2的计数系统）。表5-1是十进制数(base 10)和二进制数(base 2)的例子。位、字节和字分别是指位数不同的二进制数。

表5-1 十进制数和二进制数及他们之间的转换关系

十进制	转换		二进制
1	1=	2^0	0000 0001
2	2=	2^1	0000 0010
4	4=	2^2	0000 0100
8	8=	2^3	0000 1000
9	8+1=	2^3+2^0	0000 1001
10	8+2=	2^3+2^1	0000 1010
27	16+8+2+1=	$2^4+2^3+2^1+2^0$	0001 1011 1字节

位就是一个二进制数字（即0或1）。字节是8个二进制数字。表5-1的最后一列是字节。字是特定计算机的存储器的最小可寻址单元，它的大小由操作系统和计算机体系结构决定。计算机专家提到的“64位微处理器”和“32位操作系统”分别指计算机硬件所引用的字的大小

和操作系统所使用字的大小。字的大小会影响计算机的执行速度和性能。下面主要介绍位和字节，字将不作为重点来介绍。

有三种基本的数据类型：整数、实数和复数。这些数据集中的任意某个数都能够用一个符号表示，如，14、π和2+3i等。在计算机上计算之前，需要把这些数存入内存空间，这就需要把符号形式的数转换成一系列计算机能够识别的数值（二进制）。这种转换受到每种数值类型在计算机中所规定的存储字节数的限制。计算机使用的整数表示实际的整数，它的值有范围限制；计算机使用浮点数（*floating-point number*）表示实数，它有范围和小数位数的限制；计算机使用浮点数对来表示复数，所以它的计算规则除了复数的计算规则外，还包括浮点数的运算规则。

191

5.1.2 整数

计算机中的整数通常[⊖]用固定的二进制位数表示。本节简要介绍怎样为整数分配内存空间。由于MATLAB不使用整数变量，所以本节只是为浮点数的介绍做铺垫。

思考十进制数25是如何用二进制数表示的。首先，把十进制数用2的幂的加法表示；然后，按顺序把各个幂次表示成0、1的形式，这就是对应的二进制数。因此

$$25 = 16 + 8 + 1 = 2^4 + 2^3 + 2^0 = (11001)_2$$

符号（*bbbb*）$_2$表示基数为2的数，*b*是位值（即0或1）。内置函数`dec2bin`完成十进制到二进制的转换：

```
>> dec2bin(25)
ans =
11001
```

25需要用5位二进制数表示。表示25时并非每位上的数都为1；当5位全为1时，表示的5位数最大。与$(11111)_2$相等的十进制数为

$$(11111)_2 = 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 31$$

等价于$2^5 - 1$。出现-1是因为2^5需要6位二进制位表示。最低位对应于2^0，*n*位二进制数对应最大的十进制数为$2^n - 1$。

192

大部分计算机对整数值预备了16位（2字节）或者32位（4字节）的存储空间（单元），它们分别是整型和长整型。使用时，相同整数类型的所有整数占相同大小的内存空间。如，把十进制数123赋给16位的整型变量，所有的16位被存入内存空间，虽然只需要7位二进制数就能够表示十进制数123。这样做虽然有点浪费内存，但是由于预先知道数据的位数，计算机硬件处理起来会更简单而且高效。

绝对值最大的整数是二进制的16位数$2^{16} - 1 = 65535$。为了使计算机中能够存储负数，科学家们发明了二进制补码（*two's-complement*），它能够存储正负数，而且不需要专门的符号位。这样16位的二进制数的表示范围是[-32768，32767]。因为0也在这个范围中，所以这个范围不是对称的。

计算机可以完成从十进制数到二进制数的转换。作为计算机的使用者，我们只要记住计算机表示的整数是有范围限制的。同样，浮点数在计算机中也是有范围限制的。

⊖ 虽然在计算时可以不限制数的范围，但是大部分对整数和浮点数进行的数值计算都会限制范围和给定精度。

5.1.3 浮点数

浮点数是表示小数部分为非零数值的方法。本节介绍在浮点数二进制表示法中，浮点数的表示范围和数的精度问题。

二进制表示法 由于二进制中不存在小数点及其后面的小数部分，所以浮点数无法像整数一样存储，它要按等价的科学计数法存储。

对于一个给定的数，可以用许多等价的科学计数法表示，如

$$123.456 = 123.456 \times 10^0 = 1.23456 \times 10^2 = 123456 \times 10^{-3} = \ldots$$

上面式子的不同点在于它们的尾数中小数点的位置不同。通常我们约定尾数中小数点前一个数必须为零，小数点后一个数不能为零[⊖]。下面给出几个用这种约定方法表示的浮点数： [193]

值	规格化表示
37.5	0.3750×10^2
812.5	0.8125×10^3
0.005781	0.5781×10^{-2}

上面的这些值都可写成下列形式

$$\pm (0.d_1 d_2 \ldots d_n) \times 10^s \qquad (5\text{-}1)$$

其中d_i是尾数的十进制位数值，s是阶数。由于计算机是按二进制存储，上面方程(5-1)的等价二进制形式如下：

$$\pm p \times 2^t = \pm (b_1 b_2 \ldots b_m) \times 2^t$$

其中p是与尾数等价的二进制数，t是与阶数等价的二进制数。

单精度浮点数（*single precision*）按32位存储，*双精度浮点数*（*double precision*）按64位存储，精度用于指明每个浮点数保留多少位以及尾数和阶数各分配多少位。单精度浮点数的尾数为23位，阶数为8位；双精度浮点数的尾数为53位（包含符号位），阶数为11位（包含符号位）。双精度浮点数的等价二进制数如下所示：

64位

b *bb* ...... *bbb* *bbbbbbbbbbb*

符号位　52位尾数　11位指数，包含符号位

浮点数的有限位数表示法有下列两个主要的特点：一是浮点数的表示范围有限；二是浮点数数轴不连续。这两个特点在后面几节介绍。

范围限制 浮点数的范围由分配给存储阶数的位数所限制。双精度浮点数的阶数位数为11位，其中，符号位占一位，剩下的10位表示双精度数的阶数范围，它表示的最大十位二进制阶数是$2^{10} - 1 = 1023$。因此，双精度浮点数能表示的最大的十进制数约等于 [194]

$$2^{1023} \sim 9 \times 10^{307}$$

⊖ 请看参考文献[29]中Goldberg介绍的IEEE Floating-Point标准和参考文献[50]中Moler介绍的MATLAB怎样使用IEEE标准以获得更详细的信息。

表5-2　32位微处理器计算机上所表示的单精度数和双精度数的范围。书中常用的MATLAB变量都是双精度数。MATLAB中的复数包含双精度的实部和双精度的虚部两部分

Fortran数据类型	表示范围
整数	$-32768 \leqslant i \leqslant 32767$
单精度数	$3.40 \times 10^{38} \leqslant x \leqslant -1.18 \times 10^{-38}$
	0
	$1.18 \times 10^{-38} \leqslant x \leqslant 3.40 \times 10^{38}$
双精度数	$-1.80 \times 10^{308} \leqslant x \leqslant -2.23 \times 10^{-308}$
	0
	$2.23 \times 10^{-308} \leqslant x \leqslant 1.80 \times 10^{308}$
复数	$a+ib$，这里 $i=\sqrt{-1}$
	a和b是单精度数

表5-2对个人电脑中Fortran使用语言的数据类型的整数和浮点数的范围做了总结。在C语言中，float数据类型的数据对应Fortran语言中的单精度数，double数据类型对应Fortran中的双精度数，但是C语言中没有内置的复数数据类型。

MATLAB 第五版及更新的版本中允许创建8位的整数或者双精度型的浮点数变量。8位整数用于特定的应用中，如图像处理。为了方便，我们假设MATLAB中的数值变量都对应于Fortran中的双精度数。如果一个MATLAB变量有复数值，那么它的实部和虚部都以双精度数格式存储。

64位浮点值（双精度数）的表示范围在图5-2中以图形方式给出了。大于10^{308}的数无法用64位的浮点数表示。当计算结果出现大于10^{308}的数时，就会出现上溢错误（*overflow error*）。双精度数不仅存在上限，而且有下限。任意计算结果比10^{-308}小但严格地说又不为零的数也不能用双精度表示。即，从0到$\pm 10^{-308}$之间的数不能用双精度表示，落在数轴上的这个范围的数会出现下溢错误（*underflow error*）。

MATLAB中的内置变量realmin和realmax分别对应下溢极限（underflow limit）和上溢极限(overflow limit)。上溢极限是realmax，它把全部有效的二进制位都置为1，如果不增加位数，就不可能存储比realmax更大的数。大于realmax的数用特殊值Inf表示。

195

```
>> 10*realmax
ans =
     ∞
```

区间的另一端是realmin，它是在不丢失精度时能够存储的最小浮点数。如果把约定用于尾数的部分位数用阶数来表示，以表示那些小于realmin的数，那么这种数就叫做非规格化数（*denormal*），它在数轴上对应图5-2中 ± realmin区间上的数。当出现小于realmin的数时，可能出现两种情况。若结果比realmin稍小，就按非规格化数存储：

```
>> format long e
>> realmin/10
ans =
    2.225073858507203e-309
```

在MATLAB中，非规格化数就像其他的浮点数一样使用，但是由于它们比规格化的浮点数的有效位数少，所以它们的精度变小了。使用非规格化数允许出现渐进的下溢（gradual underflow）。当计算结果远小于realmin时，即数值太小不能用非规格化数保存时，就按零处理。

196

```
>> format long e
>> realmin/1e16
```

```
ans =
     0
```

使用非规格化数和结果归零为下溢提供了失效保护（failsafe）机制。我们谁也无法保证算法不会导致下溢，但至少出现下溢时，MATLAB不会崩溃。

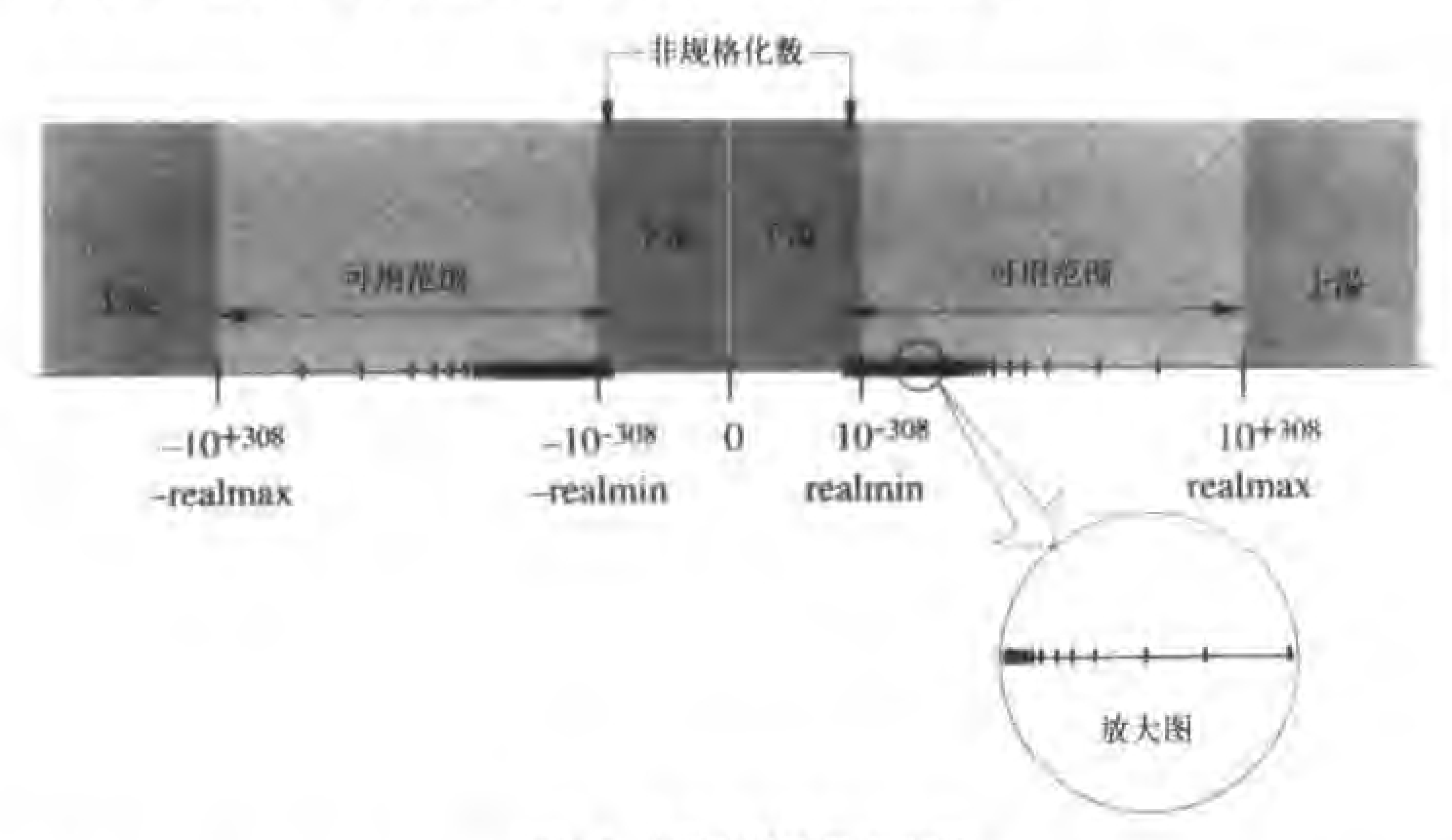


图5-2 实数轴的离散近似图

精度极限 浮点数的精度由存储尾数的位数决定。MATLAB中的双精度变量的尾数都是53位。若使用公式（5-1）的规格化形式，浮点数的尾数总是大于等于0.1小于1.0。尾数在计算机中存储时要将这些小数值表示成二进制。

把十进制小数转换成二进制数就是要把它表示成1/2的幂的形式。尾数用如下位模式计算：

$$f = \sum_{k=1}^{m} b_k 2^{-k} \tag{5-2}$$

其中b_k是与1/2的k次幂对应的位，m是计算机给尾数分配的存储位数。如，二进制尾数$(011)_2$等价于

$$0\times\left(\frac{1}{2}\right)^1+1\times\left(\frac{1}{2}\right)^2+1\times\left(\frac{1}{2}\right)^3=0.375$$

位模式中不需要小数点，因为1/2的幂自动生成了小数部分。对于整数，最高有效位在最左边，最低有效位在最右边。

直接计算公式（5-2）可以将尾数从二进制变成等价的浮点数。相反的转换并不难，假设r_k是尾数减去$\sum_{j=1}^{k} b_j 2^{-j}$后的差，即：

197

$$r_k = x-\sum_{j=1}^{k} b_k 2^{-k} = r_{k-1}-b_k 2^{-k}$$

用下列程序能够得到规格化尾数x对应的b_k序列。

算法5.1 浮点数转换成二进制数

$r_0=x$
for $k=1,2,\ldots,m$

```
    if r_{k-1} ≥ 2^{-k}
      b_k=1
      r_k=r_{k-1} - 2^{-k}
    else
      b_k=0
      r_k=r_{k-1}
    end if
  end for
```

对于十进制数0.8125，下列表格能够帮助你将十进制转换成对应的二进制数：

k	2^{-k}	b_k	$r_k=r_{k-1}-b_k 2^{-k}$
0	NA	NA	0.8125
1	0.5	1	0.3125
2	0.25	1	0.0625
3	0.125	0	0.0625
4	0.0625	1	0.0000

因此，0.8125对应的二进制数为$(1101)_2$。

由于受尾数分配的二进制位数的限制，公式（5-2）中的尾数被截断为m位，对应的十进制数为n位。这样，很多实数无法用浮点数表示。考虑分式1/5，把0.2转换成公式（5-1）对应的形式如下：

$$0011\ 0011\ 0011\ \ldots,$$

其中加入了一些空格来突出重复。十进制中的0.2能够用有限位数表示，而二进制却不能。[198] 所以，在计算机中不能用有限的位数表示十进制的0.2。这实际上是舍入误差的一个基本例子。若将尾数的位数限制为有限的二进制位（或者有限的十进制的位数），就必须把精确的实数近似为等价的浮点数。

舍入误差导致了例5.1中提到的额外数位的出现。由于0.2无法用计算机准确地表示，只能用近似值表示，在反复相加时会导致误差累积，经过3次相加后，误差就大到足够产生一个错误的十进制位。

浮点数轴　实数轴是连续的，也就是说，任意两个实数之间都存在无数的实数。如，1和2之间有1.5，1和1.5之间有1.25，如此等等。但是浮点数不是这样，下面的算法对实数进行无限循环而对浮点数只执行有限次数的循环。

算法5.2　无限二等分

```
x_1=⋯, x_2=⋯
for k=1,2,...
  δ=(x_2 - x_1)/2
  if δ=0,stop
  x_2=x_1+δ
end
```

程序清单5-1中的函数halfDiff实现了算法5.2。给函数输入参数1和2并运行，得

```
>> halfDiff(1,2)
   k        x1                          x2                          delta
   1    1.0000000000000000e+00    2.0000000000000000e+00    5.0000000000000000e-01
   2    1.0000000000000000e+00    1.5000000000000000e+00    2.5000000000000000e-01
   3    1.0000000000000000e+00    1.2500000000000000e+00    1.2500000000000000e-01
      :
  51    1.0000000000000000e+00    1.0000000000000009e+00    4.4408920985006262e-16
  52    1.0000000000000000e+00    1.0000000000000004e+00    2.2204460492503131e-16
  53    1.0000000000000000e+00    1.0000000000000002e+00    1.1102230246251565e-16
  54    1.0000000000000000e+00    1.0000000000000000e+00    0.0000000000000000e+00
ans =
    54
```

当两个浮点数的值越来越接近时，计算它们之差时，尾数部分数字的重要性越来越小了（因为这个时候浮点数的阶数已经越来越小了）。当差值小于尾数的最低有效位时，算法5.2中δ的值就为零。 199

程序清单5-1　用于等分浮点数的函数halfDiff

```
function k = halfDiff(x1,x2)
% halfDiff  Reduce the distance between two numbers until it is set to zero
%
% Synopsis:  k = halfDiff(x1,x2)
%
% Input:    x1,x2 = starting interval, only x2 is changed
%
% Output:  k = number of divisions by 2 to reduce (x2-x1)/2 to less than realmin

fprintf('   k        x1                          x2                          delta\n');
for k=1:1075
   delta = (x2 - x1)/2;
   fprintf('%4d  %24.16e  %24.16e  %24.16e\n',k,x1,x2,delta);
   if delta==0,  break,  end
   x2 = x1 + delta;
end
```

读者要区分函数halfDiff所证明的行为和下溢的概念。所有的浮点数都存在下限，函数halfDiff只是说明任意两个浮点数之间存在间隙。要弄清这个问题，读者可以自己改变函数halfDiff的输入参数（如输入100和200或－5000和+5000），运行后观察结果。

图5-2中数轴上的竖线代表浮点数。随着浮点数值的增大，两个浮点数之间的间隙（相邻的两个浮点数之间的距离）变大，相邻的两个浮点数之间间隙的大小由浮点数的尾数的最低有效位和阶数部分共同决定。

浮点数数轴的离散性对数值计算的基本操作——加法、减法、浮点数的比较——都会产生影响。当两个浮点数只有一位有效位不同时，它们相减的结果只精确到一位（小数的十进制表示而不是16进制表示对双精度数可用）。这种精度的丢失就是舍入误差。当很大的数和很小的数相加时，也会产生类似的误差。5.2节将这些结果形式化。 200

5.1.4 数值计算和符号计算

前面介绍的内容都是严格地应用于数值（*numerical*）计算（也就是说，这些计算的结果

都是数值数据）。同时，计算机也能够执行符号（*Symbolic*）计算，符号计算中，对变量操作的规则类似于人们用纸和笔作演算。可用于符号运算的商业软件包括Derive™、MACSYMA™，Maple™和Mathematica™。MATLAB附带的符号数学工具箱（*Symbolic Mathematics Toolbox*）能够执行符号运算，但是学生版的MATLAB中的该工具箱是限制版。

在符号运算中不会存在舍入误差，因为只对符号操作。赋值操作 $x=\sqrt{2}$ 将“2的平方根”和x联系在一起。相反，在数值计算中（比如MATLAB中正常的运算），`x=sqrt(2)` 把数值1.4142135...存储在分配给x的内存地址处。由于截断了后面的位数，内存中的数不精确等于 $\sqrt{2}$ ，这是由存储方案所决定的。符号运算程序能够理解 $\sqrt{2}\times\sqrt{2}$ 表示取消开根号操作，它根据符号运算的规则求出 $2=\sqrt{2}\times\sqrt{2}$ ，而不是计算1.4142135... × 1.4142135...的积。

MATLAB一般情况下所作的是数值计算。使用了符号数学工具箱后，它也能够执行符号运算。例如，计算 $\sqrt{2}\times\sqrt{2}-2$ ，输入[⊖]

```
>> sym(sqrt(2))*sym(sqrt(2)) - 2
ans =
    0
```

当执行数值计算时，显然会出现舍入误差：

```
>> sqrt(2)*sqrt(2) - 2
ans =
     4.4409e-16
```

符号运算程序使用微积分学的运算规则进行计算。如，计算下列积分

201

$$s=\int(x^2-1)\,\mathrm{d}x=\frac{x^3}{3}-x+C$$

要用到多项式的积分规则。给s赋值后，导数

$$\frac{\mathrm{d}s}{\mathrm{d}x}=x^2-1$$

就可以用另外一套规则集来计算。执行符号计算的软件包含许多代数规则以及何时和如何使用这些规则的算法。这样就不会产生舍入误差，而且函数都是连续的。

相反的是，执行数值计算的软件只用了少数的代数规则——加、减、乘、除和比较。尽管会产生舍入误差，但是速度很快。有许多问题没有分析解，只能使用数值解法，这也是数值计算的一大优点。

5.2 有限精度运算

到现在，我们已经知道了整数和浮点数在计算机内部的表示方法。下一步将介绍数值的存储情况对计算结果的影响。尽管我们能对无限位数的数值进行数学运算，但是当数值以有限数位存储在计算机内存中时，会出现舍入误差。

假设有一台理想的计算机，它能够用无限的位数表示一个数字。如果用这台计算机计算 $x=1/3$，因为x为0.3333...，所以存储x需要无限位数的内存空间。把内存中的x乘以3的结果为1.0000...，在小数点后有无限个0。这种计算就是精确算术（*exact arithmetic*），因为它不涉及到由于位数限制而产生的舍入误差。

⊖　只有安装了Symbolic Math Toolbox后这些语句才能使用。

针对计算机算术运算所涉及数据类型的不同，可以把计算机运算分成整数运算（*integer arithmetic*）和浮点运算（*floating-point arithmetic*）。整数运算能够用简单的二进制的位操作实现。只要两个整数的和或积在计算机能够表示的范围内，都能用计算机精确地表示，如下两个整数的算术运算不会有任何问题：

$$2+2=4 \quad 9\times 7=63$$

除法运算有点复杂，因为两个整数相除的商不一定是整数。一般是把所得商舍入到相邻较小的整数（向下取整）。因此： [202]

$$\frac{51}{17}=3,\quad \frac{4}{3}=1\ （整数运算）$$

显然，第二个算术运算的结果不精确。

浮点运算有自己的运算规则，它的计算结果通常情况下和精确运算的计算结果不同。在单精度浮点运算中，前面例子中的第二个商就是：

$$\frac{4}{3}=1.3333333$$

单精度数只有7位十进制位数（23bits），所以计算结果截断了后面的位数。1.3333333乘以3得到：

$$1.3333333\times 3=3.9999999$$

作为特定的工程量，3.9999999和4之间的差别并不显著，比如说一个一般用处的导管的长度，但是在有些看似无害的情况下，这种计算误差却可能导致灾难性的后果。

程序中浮点变量的精度（单精度或双精度）决定了变量存储所使用的位数。在许多计算机中，对浮点变量作数学运算时用到的数据位数要比变量实际存储的位数多（见参考文献Goldberg[29]）。中间格式不需要用户介入，计算机在使用额外位数完成计算后，会自动将结果舍入到为变量指定的精度位数。例如，在一个很长的数学公式中，乘法运算在给变量指派内存之前就进行，因计算过程中使用了额外的精度位数，所以这种情况下得到了更精确的结果。下面的MATLAB计算就是这样的：

```
>> format long e          % display all available digits
>> x = (4/3)*3
x =
     4
```

上面的计算似乎没有因为舍入而丢失精度，原因是把中间结果4/3=1.3333...存储在内部寄存器中时使用了外加的数字位数（比双精度使用的位数多），同样，1.3333... × 3的结果也用额外的数字位数存储。当对3.9999...舍入到双精度并存储时，就得到精确结果$x=4$。这种技术有助于对个别计算提供更高的精度，但是，它不能保证所有对整数的浮点运算都能够得到精确的结果。 [203]

通过保存中间计算结果，我们能够看到舍入在简单计算中的影响。参考文献[55]中Moler提出了下面示例的变种：

```
>> format long e
>> a = 4/3                  % Store double precision approx of 4/3
a =
     1.333333333333333e+00
>> b = a-1                  % Remove most significant digit
```

```
b =
    3.333333333333333e-01
>> c = 1 - 3*b                    % 3*b = 1 in exact math
c =
    2.220446049250313e-16
```

在赋值语句b=a-1中，计算所得结果尾数的位数左移了。这样，在b的最小有效位引入一个0，所以3*b的结果比1小一个最小有效位。

下一节介绍对舍入误差大小的定性度量。在介绍之前，先用下面的几个例子说明舍入误差是如何导致计算结果明显地不精确的，即使是那些简单的计算。

例5.3 二次方程求根公式中的舍入误差

我们常用的二次方程求根公式中很容易出现舍入误差，特别是在方程的系数相差很大或公式计算中使用有限的十进制位数的情况下更是如此。本例说明了误差是怎样产生的，并提出了一个更好的二次方程求根公式，这个公式用得不普遍，但是不易受舍入误差和位数限制的影响。

二次方程$ax^2+bx+c=0$的求根公式是：

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \tag{5-3}$$

由于舍入误差的存在，这个方程的解可能并不精确。考察如下实例：

$$x^2-54.32x+0.1=0 \tag{5-4}$$

它的精确解（到11位）是

$$x_1=54.318158995 \quad 、\quad x_2=0.0018410049576$$

对于上面方程，$b^2=2950.7>>4ac=0.4$。为了说明舍入误差的影响，按照4位浮点运算使用公式（5-3）计算x_1和x_2。即，计算结果只精确到4位，判别式也精确到4位。 [204]

$$\sqrt{b^2-4ac}=\sqrt{(-54.32)^2-0.4000}=\sqrt{2951-0.4000}=\sqrt{2951}=54.32$$

用$x_{1,4}$表示4位精度运算计算结果的第一个解：

$$x_{1,4}=\frac{-b+\sqrt{b^2-4ac}}{2a}=\frac{+54.32+54.32}{2.000}=\frac{108.6}{2.000}=54.30$$

x_1的误差是0.04%，这个误差不是很大。使用4位计算第二个解$x_{2,4}$为：

$$\begin{aligned}x_{2,4}=\frac{-b-\sqrt{b^2-4ac}}{2a}&=\frac{+54.32-54.32}{2.000} && \text{(i)}\\ &=\frac{0.000}{2.000} && \text{(ii)}\\ &=0 && \text{(iii)}\end{aligned}$$

这个结果的误差为100%！$x_{2,4}$的不准确近似是由于计算$\sqrt{b^2-4ac}$时的有限位舍入引起的，这导致了第（i）行中两个相等的数相减。若计算$\sqrt{b^2-4ac}$时保留更多的位数，结果$-b-\sqrt{b^2-4ac}$就不会是0。在每一步的计算中引入更多的位数能够提高结果的精度（参照习题7），但是不能完全消除两个近似值相减时由于有限精度运算导致的误差。

要解决不精确近似的问题，可以把方程（5-3）写成另一种形式，这样第二个解$x_{2,4}$受舍入

误差的影响就不会那么大了。首先，把两个求根公式的分子有理化：

$$x_1 = \frac{-b+\sqrt{b^2-4ac}}{2a}\left(\frac{-b-\sqrt{b^2-4ac}}{-b-\sqrt{b^2-4ac}}\right) = \frac{2c}{-b-\sqrt{b^2-4ac}} \tag{5-5}$$

$$x_2 = \frac{-b-\sqrt{b^2-4ac}}{2a}\left(\frac{-b+\sqrt{b^2-4ac}}{-b+\sqrt{b^2-4ac}}\right) = \frac{2c}{-b+\sqrt{b^2-4ac}} \tag{5-6}$$

现在，用公式（5-6）计算第二个解（计算 $\sqrt{b^2-4ac}$ 时，同样精确到4位）：

$$x_{2,4} = \frac{2c}{-b+\sqrt{b^2-4ac}} = \frac{0.2000}{+54.32+54.32} = \frac{0.2000}{108.6} = 0.001842$$

现在的结果只有0.05%的误差。

注意到计算$x_{2,4}$的两个公式在代数上是等价的，它们计算结果的不同仅是由舍入误差引起的。虽然公式（5-6）并不是完全不受舍入误差的影响，但是公式（5-6）不会像它所替换的公式那样产生灾难性的消去误差（cancellation error）。 205

那么我们是不是该用公式（5-5）和（5-6）代替经典求根公式？答案是否定的。用公式（5-5）计算$x_{1,4}$产生了灾难性的消去误差：

$$x_{1,4} = \frac{2c}{-b-\sqrt{b^2-4ac}} = \frac{0.2000}{+54.32-54.32} = \frac{0.2000}{0} = \infty$$

那么，我们能不能用二次方程经典求根公式计算$x_{1,4}$，并用修改后的求根公式，即公式（5-6）计算$x_{2,4}$呢？答案也是否定的。因公式的选取要视方程中系数b的符号而定，最终结果要考虑到b的符号，这样在某种程度上能防止灾难性的消去误差的发生。

建议采用下列步骤来计算二次方程的根。给定二次方程$ax^2+bx+c=0$中a、b、c的值，首先计算：

$$q \equiv -\frac{1}{2}\left[b+\operatorname{sign}(b)\sqrt{b^2-4ac}\right] \tag{5-7}$$

其中

$$\operatorname{sign}(b) = \begin{cases} 1, & \text{若 } b \geqslant 0 \\ -1, & \text{否则} \end{cases}$$

那么，二次方程的根为：

$$x_1 = \frac{q}{a}, \quad x_2 = \frac{c}{q} \tag{5-8}$$

参考文献[1、61]中介绍了上面的公式和三次方程对应的公式。练习8旨在说明公式（5-7）和（5-8）的高级舍入特性。

计算机中精确的位数多于4位，所以担心舍入误差的影响似乎是杞人忧天。使用双精度运算比单精度运算更能够减少舍入误差的影响。对于许多数值问题，特别是那些大型的线性方程组计算问题，即使是用双精度运算，某些算法还是会受舍入误差的影响，解决方法是使用那些受舍入误差的影响最小的算法。

例5.4 一个简单公式中产生巨大舍入误差的例子

可以用下列式子计算自然对数的底数

$$e = e^1 = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

206

这个极限表明随着n的增加，计算e值的精度是不确定的。虽然这种极限性质（limiting behavior）在精确算术中也成立，但是Higham指出，前面的公式在n等于某些值时很易受误差影响[36]。程序清单5-2中的函数epprox在n所列出的范围内计算$f(n) = (1+1/n)^n$并和exp(1)的值比较。运行epprox得到下列结果：

```
>> epprox
    n           f(n)              error
  1.0e+00    2.0000000000     0.7182818285
  1.0e+02    2.7048138294     0.0134679990
  1.0e+04    2.7181459268     0.0001359016
  1.0e+06    2.7182804691     0.0000013594
  1.0e+08    2.7182817983     0.0000000301
  1.0e+10    2.7182820532     0.0000002248
  1.0e+12    2.7185234960     0.0002416676
  1.0e+14    2.7161100341     0.0021717944
  1.0e+16    1.0000000000     1.7182818285
```

注意到当$n = 10^8$时，误差最小，随着n继续增加，误差变大。

程序清单5-2　函数epprox示范了在e^1近似计算中的截断误差

```
function epprox
% epprox  Demonstrate catastrophic cancellation in evaluation of
%         e = exp(1) = lim_{n->infinity} (1 + 1/n)^n
%
% Synopsis:  y = epprox
%
% Input:     none
%
% Output:    Table comparing exp(1) and f(n) = (1 + 1/n)^n as n -> infinity

%  Ref:  N.J. Higham, Accuracy and Stability of Numerical Algorithms,
%        1996, SIAM, section 1.11

e = exp(1);
fprintf('\n    n       f(n)            error\n');
for n=logspace(0,16,9)
   f = (1+1/n)^n;
   fprintf('%9.1e  %14.10f  %14.10f\n',n,f,abs(f-e));
end
```

计算$(1+1/n)^n$时，有两种误差。一种是影响相对较小的误差，另一种则是灾难性的。当n是10的次幂时，不能使用1/2的次幂来精确表示$1/n$，这时误差较小。如，把十进制数1/10转换成公式（5-1）的二进制形式所得到的是一个循环二进制数

207

00011 0011 0011 0011 ...

截断10^{-n}的二进制展开式产生的误差比计算10^{-n}产生的误差相对较小。

当n很大而且将$1/n$加上1时，即使能够用1/2的次幂精确地表示$1/n$，也会产生更大的舍入误差。当n很大时，由于操作数1和$1/n$的有效数位位置不匹配，计算$1+1/n$产生的误差相对于

$1/n$的值来说是很大的。现在介绍两个数相加时的不匹配问题。

当两个相差很大的数相加时，较小的数对最后计算结果的影响受其有效数位数的限制。假设$X = x.xxx\ldots \times 10^0$，$Y = y.yyy\ldots \times 10^{-8}$，二者都是十进制数。为了方便，假设$z = x+y<10$。写出$X$和$Y$相加的式子

```
                 可用精度
       x.xxx xxxx xxxx xxxx
  +    0.000 0000 yyyy yyyy   yyyy yyyy
  =    x.xxx xxxx zzzz zzzz   yyyy yyyy
                               丢失位
```

在本例中，数据精度为16位，Y中只有8位在结果中保留。随着X和Y之间大小数量级差距的增大，Y在计算中能用到的有效位数会减少。

对于$1+1/n$，可设X和Y分别对应1和$1/n$。当n增加时，$1+1/n$的结果接近1，但是精度降低。当$1+1/n$做n次幂运算时，误差更大。在用函数epprox计算时，在$f(10^{16}) = 1$处计算的误差达到了极点，因为用双精度浮点运算计算$1+10^{-16}$的和结果为1，这在下一节会详细介绍。

下面总结了当n很大时计算$(1+1/n)^n$产生的误差：

计算步骤	误差
$1/n$	通常$1/n$不能用1/2的次幂的有限展开式表示，因此会导致舍入误差，但是舍入误差相对于$1/n$的值来说还是很小
$1+1/n$	随着n的增加，计算的舍入误差变大，当n很大时，绝对误差相对1很小，而相对$1/n$却很大。对于$n>10^{16}$，$1+1/n$的值在双精度运算中等于1。
$(1+1/n)^n$	计算$1/n$和$1+1/n$的舍入误差被扩大。

注意到$f(n)$的误差只是由3步数学操作造成的。虽然经过数百万次的加法、乘法运算后累计的误差会导致灾难性的后果，但是本例说明了即使是少量步骤的运算也可能会导致灾难性的错误。

208

5.2.1 机器精度

对于实数轴上相近的两个数，如果它们的差比尾数的最低有效位还小，那么在浮点数轴上就不能分辨出这两个数。随着尾数位的增加，比如把单精度换成双精度，分辨不同实数的能力将会越来越强。但是，在浮点运算中，总会存在精度极限，它由一个叫做机器厄普西隆（*machine epsilon*）的数来定义，通常简称为ε_m。

机器精度也就是小数精度，它随尾数的最低有效位的改变而改变，也就是ε_m，使[⊖]

$$1.0+\delta=1.0, \quad 对任意 |\delta|<\varepsilon_m \tag{5-9}$$

成立。除非$\delta=0$时，公式（5-9）才会有数学意义，也就是说精确运算时，ε_m等价于0。因为计算机进行的是浮点运算而不是精确运算，所以$\varepsilon_m \neq 0$。在数值算法的误差分析（例见参考文献

⊖ 尾数的最低有效位的精度和公式（5-9）稍微有些不同。但是，对于理解有限精度运算的舍入误差不会产生影响。参照文献Heath [35, p.12]中此方面的例子。

[36]）中，一般更常用单位舍入数$u=\varepsilon_m/2$作为浮点数精度的量词。

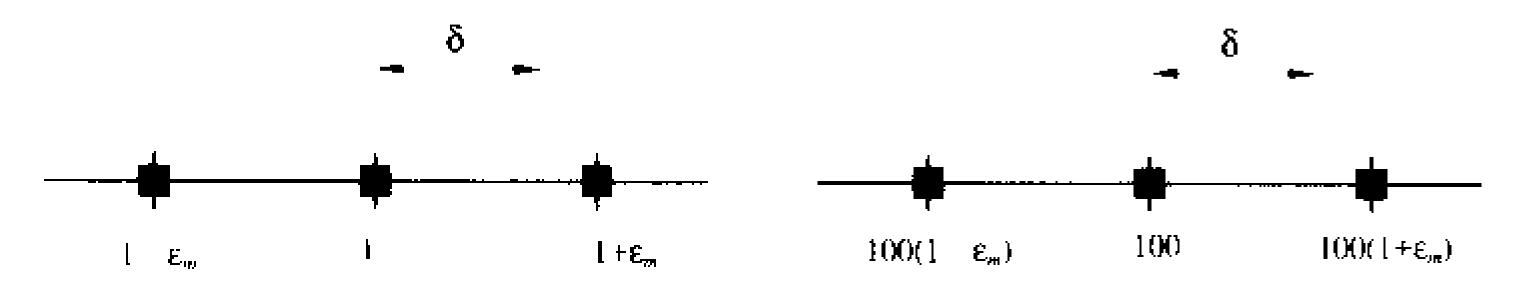


图5-3　ε_m的图形描述。实心正方形表示单个的浮点值

图5-3用图形表示了机器厄普西隆。$1-\varepsilon_m$和$1+\varepsilon_m$是最接近1的浮点数。对任意$|\delta|<\varepsilon_m$，$1+\delta$的值都会落到1和$1\pm\varepsilon_m$中间的间隙中，并且舍入为1。这个区间的长度对应于尾数最低位的精度。对于更大或更小的数量级，两个浮点数之间的距离就是ε_m的倍数。如图5-3的右边所示，最接近100的浮点数是$100(1\pm\varepsilon_m)$。

209

MATLAB的内置变量eps用于存储机器厄普西隆的值。ε_m是浮点数运算中的最细小的可分辨数值，它包含了计算中的最低位的精度值。许多内置函数把eps作为收敛公差或两个浮点数之差的最低有效位的精度。

因为很容易得到eps的值，所以不用计算ε_m。但是，学习下列计算ε_m的代码段将对读者有很大帮助：

```
epsilon = 1;
it = 0;
maxit = 100;
while it < maxit
   epsilon = epsilon/2;
   b = 1 + epsilon;
   if b == 1
      break;
   end
   it = it + 1;
end
```

本章课后练习中将深入研究本段代码。

5.2.2 程序计算中的蕴涵式

在数值计算算法设计的时候就要考虑到舍入误差带来的精度丢失。涉及舍入误差的一个典型例子就是比较两个浮点数的大小。考虑下列计算程序：

210

```
x = tan(pi/6);
y = sin(pi/6)/cos(pi/6);
if x==y
  fprintf('x and y are equal\n');
else
  fprintf('x and y are not equal:  x-y = %e\n',x-y);
end
```

输出是：

```
x and y are not equal: x-y = 1.110223e-16
```

当x和y完全相同时，判断条件if x == y才为真（也就是说，x和y的每一个二进制位都相等时）。对x和y的计算涉及到不同的数值操作，尽管$\tan(\pi/6)=\sin(\pi/6)/\cos(\pi/6)$在严格

的代数意义上成立，但是x和y仍然会相差一个很小的非零数。

上例说明了在测试两个浮点数是否相等时，会受到计算这两个数时产生的舍入误差的影响，所以，我们很少测试两个数据是否完全相等[⊖]。在处理浮点值时，通常用“x和y是否相近？”或者用“$x-y$是否足够小？”代替“x和y是否相等？”。这些测试都要求明确说明“相近”和“足够小”的含义。

5.2.3 测量误差

看看一个数α和它的近似值$\hat{\alpha}$之间的关系。既然$\hat{\alpha}$不可能与α精确地相等，只能假设它们很接近，它们的接近程度用*绝对误差*（*absolute error*）

$$E_{abs}(\hat{\alpha})=|\hat{\alpha}-\alpha| \tag{5-10}$$

或*相对误差*（*relative error*）

$$E_{rel}(\hat{\alpha})=\frac{|\hat{\alpha}-\alpha|}{|\alpha_{ref}|} \tag{5-11}$$

来测量。其中α_{ref}是一个参考值，通常，把它认为是α，所以，公式（5-11）也可以写为 [211]

$$E_{rel}(\hat{\alpha})=\frac{|\hat{\alpha}-\alpha|}{|\alpha|}$$

当α不为零时，这个公式有意义。前面的公式中，假设α的值可知。在下例之后，将讨论α是一个未知迭代序列极限时的情况。

例5.5 支票簿结算中的误差

在使用计算机进行计算时，最好能指出计算的相对误差和绝对误差。但计算相对误差比较复杂，因为有一个问题：相对于谁的误差？

假设你结算支票簿的时候发现存折上少了100美元。存根上写着上周的薪水是350美元，而且所有的收入都已存入银行的账户上，可是你的银行清单上只有250美元。在这里绝对误差和相对误差分别是：

$$E_{abs}=\$100,\qquad E_{rel}=\frac{100}{350}=0.286$$

再假设在在同一星期的同一个银行中，Megasoft公司的总裁吉尔·柏次存入她上周的薪水35000美元。但奇怪的是她的账户实际上只存入34900美元——又是100美元的误差。吉尔账户的绝对误差和相对误差分别是

$$E_{abs}=\$100,\qquad E_{rel}=\frac{100}{35000}=0.00286$$

上面两项事务的绝对误差相等，但是柏次的相对误差要少得多，这一差距是由薪水的总额多少造成的，而不是购买力的问题，因为两人分别用这100美元能够买到相同数量的咖啡、音乐CD或鞋带。

当你发现账户上少了钱后，你可能会花精力向银行提出并要求纠正这一错误。但是，柏

⊖ 在工程上最后结果的精度要求很少超过3位和4位（如指定物质的长度或被测量的值），所以当两个数x和y的前3位或4位完全相等时，它们就已足够接近了。在进行数值计算时，要达到3位和4位的精度绰绰有余，但是经常会要求更高的精度。

次很可能不会注意到这个小问题，或者即使她注意到也不会花时间向银行提出并要求纠正这一错误⊖，因为她的时间宝贵，一小时能够赚875美元。换句话说，0.286的相对误差是不能接受的，但是0.00286是微不足道的。

上例说明要正确地看待误差精度的重要性，通常，这要视情况而定。对银行来说，每周350万美元的存款出现200美元的误差是微不足道的。但是，从另一方面来说，负责的经理主管人员会针对单个存款额而不是对总存款额计算误差。

212

5.2.4 迭代序列的收敛

迭代是数值算法常用的方法。抽象地说，迭代生成一系列的标量值x_k，$k = 1, 2, 3, \ldots$当满足对所有$k > N$，有$|x_k - \xi| < \delta$，那么这个序列的极限为ξ。其中δ是一个很小的数，称为收敛公差。这样，我们称序列经过N次迭代后收敛于公差δ范围内。但是，只有满足k能够趋向无穷大时（对于收敛的迭代序列），才能知道ξ的值。在实际计算中，在k达到无穷大之前，一旦满足公差，要能够及时检测到收敛，这很重要。收敛时，必须声明x_k"足够接近"未知ξ的值。

上面的收敛条件和下式等价（见[70，5.2节]）

$$|x_i - x_k| < \delta,\ 对所有i, k > N$$

实际上，上面的式子常表示为

$$|x_{k+1} - x_k| < \delta,\ 当k > N$$

迭代次数N是否收敛受δ值的作用。当δ减小时，N增大，即达到严格的收敛需要更多的迭代次数。

例5.6　开平方根的迭代法计算

程序清单5-3中的函数newtsqrt用牛顿迭代法计算一个数的开平方根。给定数x的平方根的一个猜测值r_k，下一个猜测值是$r_{k+1} = (1/2)\ (r_k + x/r_k)$，依次继续迭代，直到$r_{k+1} \approx r_k$。程序清单5-3中的收敛判断条件并不完全，本例只是要介绍可以代替while语句中的"NOT_CONVERGED"字符串的不同表达式（注意：程序清单5-3中的代码存储在NMM工具箱中的newtsqrtBlank函数中，但是NMM中没有函数newtsqrt。本章的课后练习要求读者通过更改newtsqrtBlank.m代码来得到不同的newtsqrt函数）。

213

程序清单5-3　函数newtsqrtBlank代码包含了迭代计算$\sqrt{x}$中除收敛性判别外所有必需步骤

```
function r = newtsqrt(x,delta,maxit)
% newtsqrt   Use Newton's method to compute the square root of a number
%
% Synopsis:  r = newtsqrt(x,delta,maxit)
%
% Input:  x     = number for which the square root is desired
%         delta = (optional) convergence tolerance.  Default: delta = 5e-9
%         maxit = (optional) maximum number of iterations.  Default: maxit = 25
%
% Output:    r = square root of x to within delta/2
```

⊖ 作为一个总裁，或许相次不允许哪怕是很小的账目错误。

```
if x<0,  error('Negative input to newtsqrt not allowed');  end
if x==0,      r=x;  return;    end
if nargin<2,  delta = 5e-9;    end
if nargin<3,  maxit=25;        end

r = x/2;  rold = x;    %  Initialize, make sure convergence test fails on
                       %  first try
it = 0;
while NOT_CONVERGED & it<maxit         %  Convergence test
   rold = r;                           %  Save old value for next convergence test
   r = 0.5*(rold + x/rold);            %  Update the guess
   it = it + 1;
end
```

程序清单5-4中的函数testsqrt调用函数newtsqrt时输入了一连串的值，并比较了使用函数newtsqrt与内置函数sqrt的精确性。编写出函数newtsqrt后，读者可以调用函数testsqrt。

214

程序清单5-4　函数testsqrt使用一系列输入值测试newtsqrt函数（注意：函数newtsqrt不是工具箱NMM中的函数。要运行testsqrt，需要打开newtsqrtBlank.m文件后适当地更改收敛条件，并将结果函数保存为newtsqrt函数。）

```
function testSqrt
% testSqrt  Test the newtsqrt function for a range of inputs

xtest = [4  0.04  4e-4  4e-6  4e-8  4e-10  4e-12];   % arguments to test

fprintf('\n Absolute Convergence Criterion\n');
fprintf('    x          sqrt(x)    newtsqrt(x)    error       relerr\n');

for x=xtest          %  repeat for each column in xtest
   r = sqrt(x);
   rn = newtsqrt(x);
   err = abs(rn - r);
   relerr = err/r;
   fprintf('%10.3e  %10.3e  %10.3e  %10.3e  %10.3e\n',x,r,rn,err,relerr)
end
```

下列代码段对收敛判断都不够准确：

```
while r~=rold & it<maxit              %  Bad test #1


while (r-rold)>delta & it<maxit       %  Bad test #2
```

第一个条件要求r和rold完全相等，这只有当r和rold的每一位都相等时才能满足。这表明由计算步骤产生的r和rold相差不到ε_m。第二个条件只要r大于rold而不是它们的值相近时就满足。下列的代码段能明显地改进上面的测试：

```
while abs(r-rold)>delta & it<maxit     % absolute difference
```

当函数newtsqrt采用绝对偏差的收敛性判定准则时，函数testsqrt的输出结果为：

```
>> testsqrt      %  absolute difference convergence criterion
```

```
     x          sqrt(x)    newtsqrt(x)    error       relerr
4.000e+00     2.000e+00    2.000e+00    0.000e+00    0.000e+00
4.000e-02     2.000e-01    2.000e-01    0.000e+00    0.000e+00
4.000e-04     2.000e-02    2.000e-02    0.000e+00    0.000e+00
4.000e-06     2.000e-03    2.000e-03    2.342e-17    1.171e-14
4.000e-08     2.000e-04    2.000e-04    1.649e-15    8.246e-12
4.000e-10     2.000e-05    2.000e-10    2.000e-05    1.000e+00
4.000e-12     2.000e-06    2.000e-12    2.000e-06    1.000e+00
```

当函数newtsqrt中的自变量较小时，绝对偏差收敛性准则就会失效。例如，设$x = 4 \times 10^{-10}$，只有当相对误差为100%时才能达到所定义的绝对误差公差。在收敛判别里使用相对偏差（*relative difference*）能够提高函数newtsqrt的可靠性：

215

```
while abs((r-rold)/rold)>delta & it<maxit     % relative difference
```

采用相对偏差作为收敛性判定准则后，testsqrt的输出为：

```
>> testsqrt     %  relative difference convergence criterion

     x          sqrt(x)    newtsqrt(x)    error       relerr
4.000e+00     2.000e+00    2.000e+00    0.000e+00    0.000e+00
4.000e-02     2.000e-01    2.000e-01    0.000e+00    0.000e+00
4.000e-04     2.000e-02    2.000e-02    0.000e+00    0.000e+00
4.000e-06     2.000e-03    2.000e-03    0.000e+00    0.000e+00
4.000e-08     2.000e-04    2.000e-04    2.711e-20    1.355e-16
4.000e-10     2.000e-05    2.000e-05    3.388e-21    1.694e-16
4.000e-12     2.000e-06    2.000e-06    0.000e+00    0.000e+00
```

相对偏差准则估量了前后r值的变化。只要rold ≠ 0，此判别方法就适用于任何不受输入参数影响的收敛公差。

5.2.5 相对收敛性准则和绝对收敛性准则

上面的两个例题介绍了两种收敛性准则

$$|x_k - x_{k-1}| < \Delta_a, \quad \left|\frac{x_k - x_{k-1}}{x_{k-1}}\right| < \delta_r \tag{5-12}$$

其中Δ_a和δ_r分别代表绝对收敛公差和相对收敛公差。如果问题适当地缩放，相对误差不受影响，因此一般更常用相对收敛性准则。例如，若计算一个人体重为60公斤时产生0.5公斤的误差，其相对误差是0.83%。这个相对误差不随体重的单位而变化，不管是英镑、吨还是公吨，结果都一样。

相对差值收敛性准则中，δ_r可以视为计算结果的可接受最高偏差。所以，通常把相对收敛公差定义为5×10^{-p}，允许结果的第$(p-1)$位的值不确定。例如，值$\delta_r = 5 \times 10^{-4}$说明r和rold的差值要小于rold的第3位数。因子5作为上舍和下舍的标准，大于5的数上舍，小于5的数下舍。1.234567保留两位的结果是1.2，因为3<5，而保留6位的结果是1.23457，因为6>5。

当级数的极限为零时，使用绝对收敛性准则更好。还有一种收敛性准则结合了绝对收敛准则和相对收敛准则两者的优点：

$$|x_k - x_{k-1}| < \max[\Delta_a, \delta_r |x_{k-1}|] \tag{5-13}$$

216

使用这种复合准则，迭代要一直进行到满足最大公差为止。

尽管Δ_a和δ_r的值随具体问题而不同，但是ε_m是它们共同的下界。当$\Delta_a < \varepsilon_m$或$\delta_r < \varepsilon_m$时，使用

公式（5-12）和（5-13）会引起无限循环。由于舍入误差的出现，即使一个迭代序列在精确算术运算中是收敛的，但是x_k和x_{k-1}之差可能永远都不会比ε_m小。这在实际中要灵活应用（如，编写一个通用程序的时候，$5\varepsilon_m$、$100\varepsilon_m$或更大的偏差很可能远远超出了大多数工程应用对误差容限的需求）。

5.3 算法的截断误差

*截断误差(truncation error)*是由于用离散、代数的公式来近似连续的数学表达式时产生的。与舍入误差不同的是，截断误差不受硬件和计算机语言的影响，而是受程序员或用户控制的。使用更精确的离散逼近能够减小截断误差，但是却不能消除截断误差。

不同的数值方法存在不同的截断误差分析方法。本书针对具体的数值方法介绍不同的截断误差特性。本节将通过一个无穷级数介绍截断误差的定义，下节将结合函数的泰勒级数展开式介绍截断误差。

思考e^x的展开式：

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \tag{5-14}$$

不可能也没必要通过计算e^x的每一个展开项来求和。在某个位置可以忽略高次项，因为它们不会对最后的累加结果有太大的影响。由有限个展开项相加计算出来的值和e^x的精确值之间的差就称为截断误差，截断二字说明了误差产生的原因，即截断误差是由级数的截断产生的。公式（5-14）中的截断误差也受函数e^x自变量的影响。当$x<<1$时，级数项的分子衰减很快，而对于较大的x，则需要很多项之后分母的阶乘值才能超过分子求幂后的值。

计算e^x精确值时，所需的项数与计算每一项所用的有效位数无关。即，舍入误差和截断误差是独立的互不影响的，例5.8说明了这一点。

例5.7 计算sin(x)的级数

函数sine用下列无穷级数定义

217

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

虽然随着次数的增加，项的值越来越小，但是后续项的分子和分母值都在急剧地增加。使用递归方法可以避免某一项的值溢出，若T_k是第k项（$k = \{1, 3, 5, \ldots\}$）那么

$$T_k = \frac{-x^2}{k(k-1)} T_{k-2}$$

由于每一项的分子和分母都是在前一项的基础上得到的，所以避免了溢出和精度丢失，而且还提高了计算的效率。

程序清单5-5中的函数`sinser`使用了递归公式和下列终止准则

$$\left|\frac{T_k}{S_k}\right| < \delta$$

其中S_k是前k项的累加和。练习22将会让你探究累加计算终止的准则。函数`sinser`运行后的典型结果如下所示：

```
>> sinser(pi/6);
Series approximation to sin(0.523599)
```

```
 k      term          ssum
 1    5.236e-01    0.52359878
 3   -2.392e-02    0.49967418
 5    3.280e-04    0.50000213
 7   -2.141e-06    0.49999999
 9    8.151e-09    0.50000000
11   -2.032e-11    0.50000000

Truncation error after 6 terms is  3.56e-14
```

改变函数sinser的参数*tol*（可选）和*nmax*的值能够进一步地研究截断误差的变化特性。sine函数的截断误差受参数的影响也很大。例如，这可通过函数sinser分别计算$x=\pi/6$、$5\pi/6$、$11\pi/6$、$17\pi/6$时的sin（x）值来得到些启发。思考一下，如何修改函数sinser才能够使它避免由这些输入参数带来的意外变化?

程序清单5-5　函数sinser使用截断级数的办法来计算sin(x)的值

```
function ssum = sinser(x,tol,nmax)
% sinser  Evaluate the series representation of the sine function
%
% Synopsis: ssum = sinser(x)
%           ssum = sinser(x,tol)
%           ssum = sinser(x,tol,nmax)
%
% Input:    x   = argument of the sine function, i.e., compute sin(x)
%           tol = (optional) tolerance on accumulated sum. Default:  tol = 5e-9
%                 Series is terminated when abs(T_k/S_k) < delta.   T_k is the
%                 kth term and S_k is the sum after the kth term is added.
%           nmax = (optional) maximum number of terms. Default: n = 15
%
% Output:   ssum = value of series sum after nterms or tolerance is met

if nargin<2,  tol = 5e-9;  end
if nargin<3,  nmax = 15;   end

term = x;  ssum = term;          %  Initialize series
fprintf('Series approximation to sin(%f)\n\n  k       term          ssum\n',x);
fprintf('%3d  %11.3e  %12.8f\n',1,term,ssum);

for k=3:2:(2*nmax-1)
  term = -term * x*x/(k*(k-1));                   %  Next term in the series
  ssum = ssum + term;
  fprintf('%3d  %11.3e  %12.8f\n',k,term,ssum);
  if abs(term/ssum)<tol, break;  end              %  True at convergence
end
fprintf('\nTruncation error after %d terms is %g\n\n',(k+1)/2,abs(ssum-sin(x)));
```

例5.8　e^x级数的舍入误差和截断误差

218 / 219

公式（5-14）给出了e^x的级数展开式。令T_k为级数的第k项，S_k为前k项和：

$$T_k = \frac{x^k}{k!}, \qquad S_k = 1 + \sum_{j=1}^{k} T_k$$

若从第k项开始截断，则级数近似值的绝对误差为

$$E_{abs,k}=|S_k - e^x|$$

程序清单5-6中的函数expSeriesPlot计算了前k项的和，并且画出了$E_{abs,k}$ 对k的函数图。和例5.7一样，这里也使用递归法计算。图5-4画出了用语句

```
>> expSeriesPlot(-10,5e-12,60)
```

计算exp(- 10)的绝对误差函数图。当$|x|$>> 1时，绝对误差一开始会增加，因为分子求幂得到的值比分母的阶乘项增加速度快。从$x = -10$，k>10起，分母的阶乘开始决定展开项值的大小。当k增加时，截断误差减小，于是$E_{abs,k}$减小。

最后，舍入使S_k不再变化。当$T_{k+1}\to 0$时，语句

```
ssum = ssum + term
```

不会改变ssum的值了。当$x = -10$时，这种现象开始于$k \sim 48$处。此时的截断误差$|S_k - e^x|$不为零，但是却有$|T_{k+1}/S_k|<\varepsilon_m$。本例说明截断误差和舍入误差二者是独立不相干的。当k<48时，截断误差在级数的计算误差中占主要地位；而当k>48时，由于舍入误差，截断误差不会一直减小下去。

程序清单5-6 计算e^x的级数并画出绝对误差作为累加和项数的函数图形

```
function ssum = expSeriesPlot(x,tol,n)
% expSeriesPlot  Evaluate and plot series representation of exp(x)
%
% Synopsis:  ssum = expSeriesPlot(x)
%            ssum = expSeriesPlot(x,tol)
%            ssum = expSeriesPlot(x,tol,n)
%
% Input:  x   = argument of the exp function, i.e., compute exp(x)
%         tol = (optional) tolerance on accumulated sum. Default: tol = 5e-9
%               Series is terminated when T_k/S_k < delta, where T_k is the
%               kth term and S_k is the sum after the kth term is added.
%         n   = (optional) maximum number of terms. Default: n = 15
%
% Output:   ssum = value of series sum after n or tolerance is met

if nargin < 2,  tol = 5e-9;  end
if nargin < 3,  n = 15;      end

term = 1;  ssum = term;  Eabs(1) = abs(ssum-exp(x));   % Initialize
fprintf('Series approximation to exp(%f)\n\n',x);
fprintf('  k      term        ssum        Eabs\n');
fprintf('%3d  %11.3e  %11.3e  %11.3e\n',1,x,ssum,Eabs(1));

for k=2:n
  term = term*x/(k-1);                % Next term in the series
  ssum = ssum + term;
  Eabs(k) = abs(ssum-exp(x));
  fprintf('%3d  %11.3e  %11.3e  %11.3e\n',k,term,ssum,Eabs(k));
  if abs(term/ssum)<tol, break;  end    %  True at convergence
end

semilogy(1:k,Eabs,'-');
xlabel('Number of terms');   ylabel('Absolute Error');
fprintf('\nTruncation error after %d terms is %11.3e\n\n',k,Eabs(k));
```

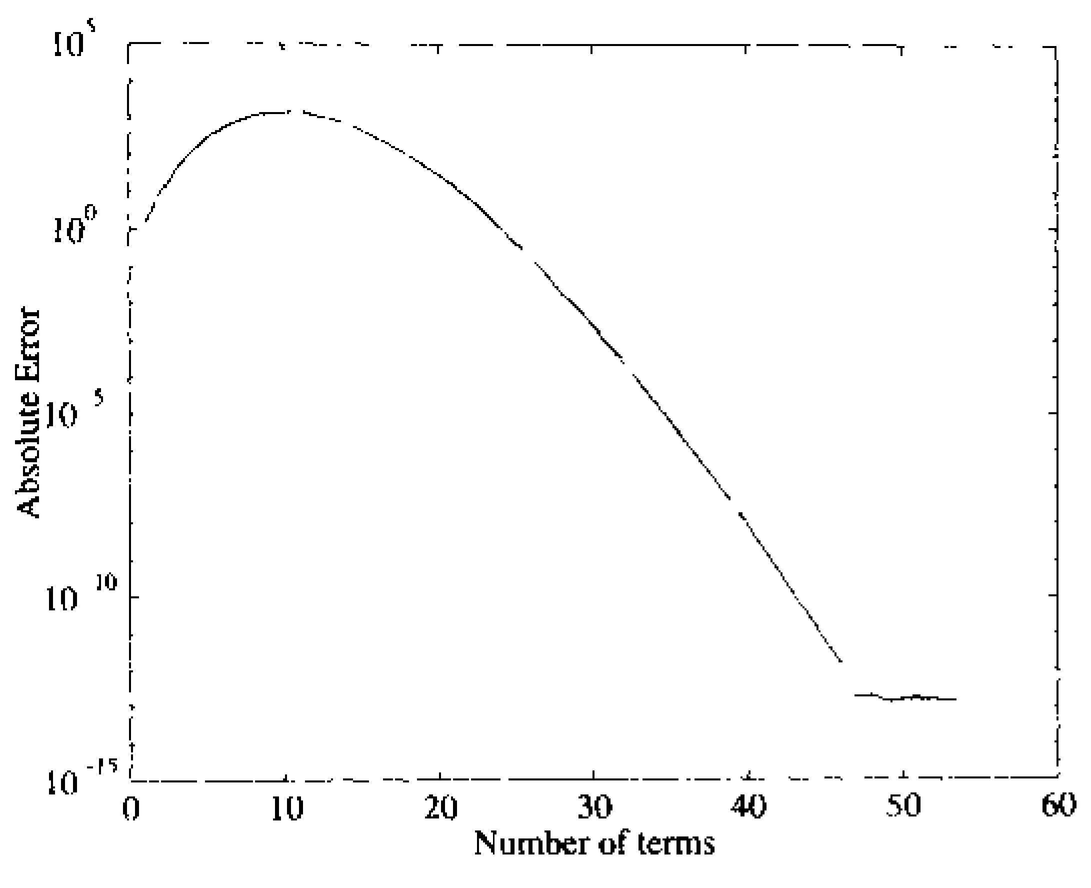


图5-4 e^x的级数近似值图（$x=-10$）

5.3.1 泰勒级数

泰勒级数近似一般用来分析和研究数值方法的特性。本节定义了泰勒级数的基本属性，并通过例子介绍了如何利用泰勒级数来描述数值计算中的截断误差。

对于$x \in [a, b]$区间上的完全可微分（sufficiently differentiable）函数$f(x)$，参照文献[9]假设$P_n(x)$作如下定义：

$$P_n(x)= f(x_0)+(x-x_0)\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_0}+\frac{(x-x_0)^2}{2}\left.\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right|_{x=x_0}+\cdots+\frac{(x-x_0)^n}{n!}\left.\frac{\mathrm{d}^n f}{\mathrm{d}x^n}\right|_{x=x_0} \tag{5-15}$$

那么，存在$\xi(x)$，$x_0\leqslant\xi(x)\leqslant x$，满足

221

$$f(x)=P_n(x)+R_n(x) \tag{5-16}$$

而且

$$R_n(x)=\frac{(x-x_0)^{(n+1)}}{(n+1)!}\left.\frac{\mathrm{d}^{(n+1)}f}{\mathrm{d}x^{(n+1)}}\right|_{x=\xi} \tag{5-17}$$

多项式$P_n(x)$称为$f(x)$的n阶泰勒级数近似。当$P_n(x)$是对$f(x)$的近似时，$R_n(x)$为余数或截断误差。

公式（5-17）对$R_n(x)$的定义说明截断误差是能够计算的，但是通常情况下，由于ξ不可知，所以无法计算。这点缺陷并不影响大局，因为选取一种算法最重要的就是了解截断误差的一般特性而不是它的精确值。

实际上，把公式（5-17）中的导数项视为未知常量，主要关心的是系数$(x-x_0)^{(n+1)}/(n+1)!$。通常我们定义$h=x-x_0$，且一般$h<<1$。这样，当n增加时，$h^{n+1}/(n+1)!$会很快地趋向于零。只要f的导数有上下限，当n增加时，有$R_n\to 0$。这时，就需要程序员、分析员或者用户（随机应变地）选择n值使截断误差落在可接受的公差内。

例5.9 用泰勒级数近似1/（1－x）

思考下列函数

$$f(x)=\frac{1}{1-x} \tag{5-18}$$

对函数$f(x)$的1阶、2阶和3阶泰勒级数近似分别为

$$P_1(x)=\frac{1}{1-x_0}+\frac{x-x_0}{(1-x_0)^2} \tag{5-19}$$

$$P_2(x)=\frac{1}{1-x_0}+\frac{x-x_0}{(1-x_0)^2}+\frac{(x-x_0)^2}{(1-x_0)^3} \tag{5-20}$$

$$P_3(x)=\frac{1}{1-x_0}+\frac{x-x_0}{(1-x_0)^2}+\frac{(x-x_0)^2}{(1-x_0)^3}+\frac{(x-x_0)^3}{(1-x_0)^4} \tag{5-21}$$

其中x_0是泰勒级数的展开点。程序清单5-7中的函数demoTaylor计算并画出了在用户定义的邻域内这些近似函数的图形。图5-5画出了在区间$1.2<x<2$的范围内这些近似值对应的图形。使用语句

222

```
>> demoTaylor(1.6,0.8)
```

可以得到这些图形。在$x_0=1.6$的附近所有的泰勒多项式与$f(x)$一致。多项式的次数越高，产生这种一致的x的范围就越大。

注意到公式（5-20）和（5-21）并没有比原始函数（5-18）节省多少计算量。本例的目的在于介绍如何用泰勒级数近似某个函数。图5-5表明，使用泰勒级数的项数越多，最终结果就越接近原始的函数。

程序清单5-7 函数demoTaylor计算并画出$f(x)=1/(1-x)$的泰勒级数近似值

```
function demoTaylor(x0,dx)
% demoTaylor  Taylor Series approximations for f(x) = 1/(1-x)
%
% Synopsis:  demoTaylor
%            demoTaylor(x0,dx)
%
% Input:  x0 = (optional) point about which the Taylor Series expansion is
%              made.  Default:  x0 = 1.6;
%         dx = (optional) size of neighborhood over which the expansion
%              is evaluated.  Default:  dx = 0.8
%
% Output:  a plot of f(x) and its Taylor Series approximations

if nargin<2,  x0 = 1.6;  dx = 0.8;  end

x = linspace(x0-dx/2,x0+dx/2,20);   % x-values at which f(x) is evaluated
fx = 1./(1-x);                      % Exact f(x); notice the array operator

h = x - x0;                      % Avoid recomputing intermediate values,
t = 1/(1-x0);                    % h and t
p1x = t*ones(size(x)) + h*t^2;   % First order Taylor polynomial
p2x = p1x + (h.^2)*t^3;          % Second order "  "  "
p3x = p2x + (h.^3)*t^4;          % Third
```

```
plot(x,fx,'-',x,p1x,'o-',x,p2x,'^-',x,p3x,'s-');
legend('exact  ','P_1(x)','P_2(x)','P_3(x)',4);
xlabel('x');    ylabel('Approximations to f(x) = 1/(1-x)');
```

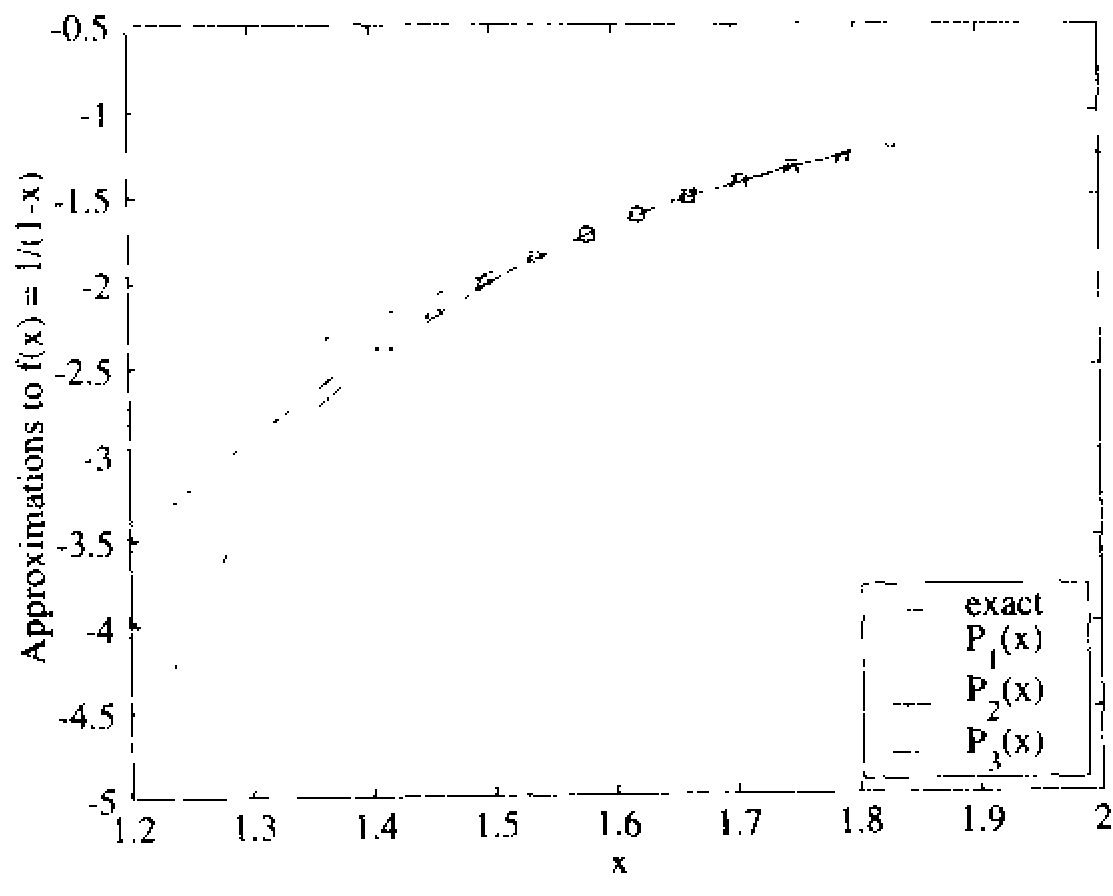


图5-5　$f(x)=1/(1-x)$的泰勒级数近似值的图形

5.3.2 阶符

数量级估算是精确数值的重要补充。所谓的经验法则（*rule of thumb*）或封底计算（*back-of-the-envelope calculation*）是使用数量级估算和简单算术运算求出结果，所得结果通常精确到小数点后一位。这种估算有助于对一个设计或计算程序进行可行性评估。只有经验丰富的工程师或分析员才能够正确地使用数量级估算法。

[223]

简单数量级估算　思考一下进行大型矩阵运算所需内存。$n \times n$的矩阵需要存储n^2个浮点数。在MATLAB中，矩阵的每个元素占8个字节内存。因此，一个100 × 100的矩阵需要的内存空间至少为$8 \times 100^2 = 8 \times 10^4 = 0.08$兆字节，分配这么小的内存不存在任何问题。前一语句中的"至少"是说，真正计算时还会使用到一些额外量（如矩阵计算中用到的向量），它们也需要内存空间。继续进行内存估算，1000 × 1000的矩阵至少要8兆字节，而10000 × 10000的矩阵至少要800兆字节。有了这些信息，我们就可以很快地确定出计算机能否分配足够的内存进行计算。这种估算的好处是不用编程或分析就可以很快得出结果。要注意矩阵运算时内存限制并非是可行性的惟一限制，算法的运行时间和矩阵大小也是密切相关的，对许多涉及矩阵的重要运算，计算代价会以n^3的速度增加（详见第8章）。

[224]

阶符（*order notation*）可以用于简化数量级计算，它用于表达数学公式中的基项（dominant term）的数量级。某个数的数量级用大写字母"O"表示，例如，MATLAB中存储一个$n \times n$矩阵所需要的内存是$8n^2$，或者$O(n^2)$，可以读作"大O　n的平方"或"n平方的阶"。一般情况下，常数因子在数量级估算时可以省略，像前面提到的系数8。尽管这些因子会影响到最后的数量级估算的结果，但是同一计算中的任意两个数量级估算中这些因子都相同，最

后比较时都会消去。例如，前面对内存的估算中，1000 × 1000的矩阵是100 × 100矩阵所占内存的100倍，这与因子8没有关系。

对某个数进行数量级估算时，只保留表达式中的基项。例如，公式（5-16）和（5-17）中的泰勒级数可以写成简便形式

$$f(x)=P_n(x)+\mathcal{O}\left(\frac{(x-x_0)^{n+1}}{(n+1)!}\right)$$

或

$$f(x)=P_n(x)+\mathcal{O}(h^{n+1}) \tag{5-22}$$

其中$h=x-x_0$。因子$1/(n+1)!$已舍弃，因为对于很小的h来说，随着n增加h^{n+1}比$1/(1+n)!$接近零的速度更快，即，h^{n+1}对$h^{n+1}/(n+1)!$的数量级的影响更大，所以在数量级估算时只保留了h^{n+1}。

阶符有助于灵活地书写数学表达式。公式（5-22）并不是说h^{n+1}加上$P_n(x)$以使等式成立，它只是说明$f(x)$和$P_n(x)$之间相差一个很小的数h^{n+1}，而且，随着n的增加这一差值会越来越小。

例5.10 e^x的近似级数的阶

e^x的无穷级数表达式为

$$e^x=1+x+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}+\cdots$$

若$x<1$，式中后面的每一项比前一项的数值更小。在级数的有限项终止时将导致高次项的丢弃，例如，近似

$$e^x\approx 1+x+\frac{x^2}{2!}+\mathcal{O}(x^3)$$

$$e^x=1+x+\frac{x^2}{2!}+\cdots+\frac{x^n}{n!}+\mathcal{O}(x^{n+1})$$

的误差阶数分别是x^3和x^{n+1}。当报告这些近似级数的阶数时，与x^3和x^{n+1}相乘的常量要省略。

误差分析中更精确的阶符 推导某个算法的截断误差时，有时需要更精确的阶符。推导完成后，可以用阶符来使用和报告截断误差的大小

考虑精确量函数$f(x)$的数值近似表达式$F(x, h)$。$F(x, h)$的误差由x的值和算法所用的参数h决定。因h只是算法的一个参数，所以它不对$f(x)$的真实值产生任何影响。根据定义，$f(x)$和$F(x, h)$之间的差就是误差$E(x, h)$：

$$f(x)=F(x,h)+E(x,h) \tag{5-23}$$

通常，由于$f(x)$未知，所以$E(x, h)$也未知；否则，就没有必要编写计算$F(x, h)$的程序。通常情况下，像公式（5-22）中介绍的，h用于定义$F(x, h)$近似值的项与项之间的步长（step size）。在精确的误差分析中，如公式（5-23），如果

$$|E(h,x)|\leqslant C(x)h^{\alpha} \tag{5-24}$$

就说其中的$E(x, h)$为$O(h^{\alpha})$，其中因子$C(x)$（也可能为常量）和h值无关。这时，可以说计算$f(x)$的近似值$F(x, h)$的算法是α阶的。

用截断误差的阶数作为调试的手段 第4章介绍了调试数值算法用到的程序，这里将给出这种工具的简单数学背景。

225

假设编写好了计算$f(x)$的近似值$F(x,h)$的程序，程序的截断误差形式如公式（5-24）。为了测试程序及程序的实现，把它用在一个有已知解的问题中（即已知函数$f(x)$）。其误差为：

$$E(x,h)=F(x,h)-f(x)$$

226

现在，假设用不同的h值分别计算$F(x,h)$，根据公式（5-24），这些误差的比率为

$$\frac{E(x,h_2)}{E(x,h_1)}=\left(\frac{h_2}{h_1}\right)^{\alpha}$$

或

$$\alpha=\frac{\log\left(\frac{E(x,h_2)}{E(x,h_1)}\right)}{\log\left(\frac{h_2}{h_1}\right)} \tag{5-25}$$

由公式（5-25），我们可以用数值实验来测量α的值。若程序中正确地实现了算法，则所测量的α值应该和通过分析方法得到的α值一致。下面的例子用简便形式示范了这种技术。

例5.11 求圆的内接多边形的周长问题

Kahaner 等人提出了计算π的阿基米德（Archimedes）算法的变体，见文献[4，p.38]。在阿基米德算法中是通过计算直径已知的圆的一系列内接正多边形的周长来计算π值的。图5-6中分别画出了正方形、八边形和十六边形。本算法介绍了在一个已知解的问题中应用数值程序时如何计算截断误差的阶数α。

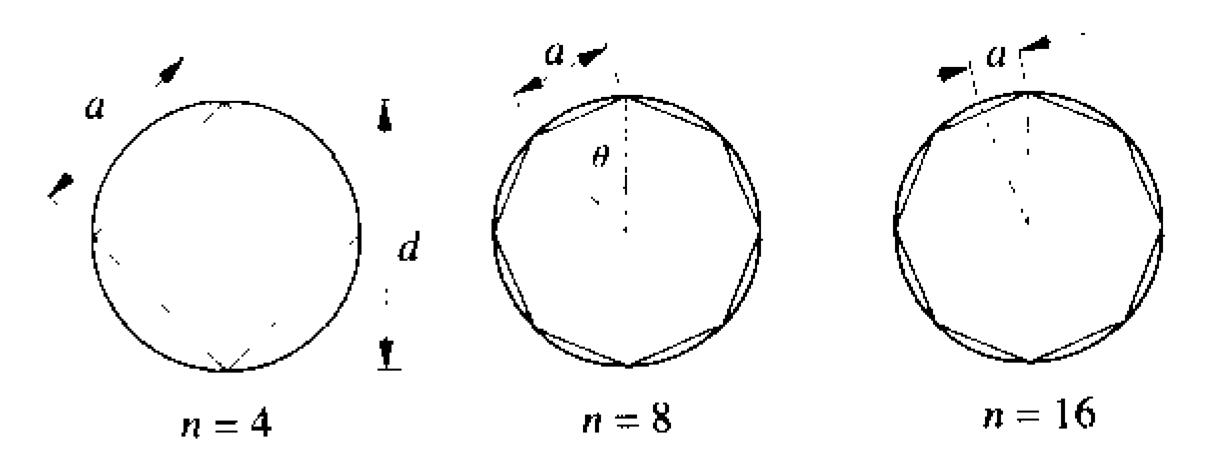


图5-6 利用圆内接多边形计算π值

直径为d的圆其内接正n边形的边长a可通过做简单的几何计算得到：

$$a=d\sin\frac{\theta}{2}$$

对应弦长a的夹角θ可以通过$\theta=2\pi/n$求得，多边形的周长p为na，即

227

$$p=nd\ \sin\frac{\pi}{n}$$

假设用正n边形的周长来近似圆的周长，其误差为

$$E(d,n)=d\left(n\ \sin\frac{\pi}{n}-\pi\right)$$

注意到$\pi/n\rightarrow 0$时，$\sin(\pi/n)\rightarrow\pi/n$而且$E\rightarrow 0$。

进行实例计算的目的是想让读者看看E会以多快的速度趋向于0。为方便起见，设$h = 1/n$。那么按照前一节介绍的表示法，有$E(d, h) = O(h^{\alpha})$，其中α未知。程序清单5-8中的函数Archimedes用公式（5-24）来得到α的值。运行Archimedes函数的结果如下：

```
>> Archimedes
   n      difference     alpha
   4    -0.313165529
   8    -0.080125195    1.96660
  16    -0.020147501    1.99166
  32    -0.005044163    1.99791
  64    -0.001261497    1.99948
```

当n增加时，α的值趋近2，因此，$E(d, h) = O(h^2) = O(1/n^2)$，这和截断误差的理论分析结果一致，练习29也能够得到相同的结果。

程序清单5-8 计算圆周长的函数Archimedes。用圆内接n边形的周长近似为圆周长

```
function Archimedes(d)
% Archimedes  Perimeter of an n-sided polygon inscribed in a circle.
%
% Synopsis:   Archimedes
%             Archimedes(d)
%
% Input:      d = (optional) diameter of the circle;  Default:  d = 1
%
% Output:     Differences between circumference, c, of a circle and perimeter,
%             p, of an n-sided polygon inscribed in the circle.

if nargin<1; d = 1;  end

fprintf('   n      difference     alpha\n');
for n = [4 8 16 32 64]
  p = n*d*sin(pi/n);
  dif = p - pi*d;
  fprintf(' %3d   %12.9f',n,dif);
  h = 1/n;
  if n>4         %  compute alpha only if there is enough data
    fprintf('   %8.5f\n',log(dif/difold)/log(h/hhold));
  else
    fprintf('\n');
  end
  difold = dif;   hhold = h;          %  save for next iteration
end
```

228

例5.12 用有限差分法求舍入误差和截断误差

在许多应用中，函数的导数也需要采用近似计算的办法才能得到。当函数未知时，例如当微分方程的解函数没有解析解时，那么就无法用分析方法对这些函数求微分。有限差分（*finite-difference*）近似利用函数$f(x)$在邻近点范围内的离散值计算该点的导数值。本例就是用有限差分法来近似已知函数的一阶导数并把所得结果和函数的精确值进行比较。

函数$f(x)$在$x+h$处的泰勒级数近似如下所示，这里h是个很小的距离，称为步长。将公式（5-15）整理成关于h的形式：

$$f(x+h)=f(x)+hf'(x)+\frac{h^2}{2}f''(x)+\cdots$$

其中 $f'(x)\equiv df/dx$、$f''(x)\equiv d^2f/dx^2$。由上面公式解出 $f'(x)$ 得:

$$f'(x)=\frac{f(x+h)-f(x)}{h}-\frac{h}{2}f''(x)+\cdots$$

根据上述计算我们得到，当h趋于0时，公式右边的第一项就是函数一阶导数的精确值。对于有限值h，导数的有限差分近似为:

$$f'_{fd}(x)=\frac{f(x+h)-f(x)}{h}+\mathcal{O}(h) \tag{5-26}$$

计算f'_{fd}时，可以忽略$O(h)$项。

截断误差的最重要的一项是$(h/2)f''(x)$，其中$f''(x)$也未知。但只要$f''(x)$有界（也就是说，存在常量C，对任意$x\le\xi\le x+h$，满足$f''(\xi)\le C$），$f''(x)$的真实值并不重要，但是我们不能任意地改变它的值。h的值可由我们选择，截断误差$O(h)$中的h是惟一一个我们可以控制的量。

[229]

有限差分近似用于求解微分方程的数值解。本例中，我们只关心公式（5-26）对一个已知方程的一阶导数的近似程度。为简单起见，假设$f(x)=e^x$。理论上的一阶导数为$f'(x)=e^x$。有限差分近似的相对误差为

$$E_{rel}=\frac{f'_{fd}(x)-f'(x)}{f'(x)}=\frac{f'_{fd}(x)-e^x}{e^x}$$

相对误差依赖于h、x，其中x是求导的那一点。

程序清单5-9 函数fidiff示范了舍入和截断误差对$x=1$处的d(e^x)/dx进行有限差分近似的作用

```
function  fidiff(x)
% fidiff  First order finite-difference approximation to d/dx of exp(x)
%
% Synopsis:  fidiff(x)
%
% Input:     x = value at which the derivative is to be evaluated
%
% Output:    A table and plot so show how the error varies with stepsize

fp = exp(x);                        % Exact value of fprime
h = logspace(-12,0,13)';            % Column vector of stepsizes
fpfd = (exp(x+h) - exp(x))./h;      % Vectorized calculation of fd approximation
Erel = abs(fpfd - fp)./fp;          % and relative errors

fprintf('     h        fp        fpfd          Erel\n');
for k=1:length(h)
  fprintf('%10.1e %9.5f %9.5f %12.2e\n',h(k),fp,fpfd(k),Erel(k));
end

loglog(h,Erel,'+')
xlabel('Stepsize,  h');    ylabel('Relative error');
```

程序清单5-9中的函数fidiff计算了e^x的一阶导数的精确值和有限差分近似值。给定x的一个输入值，对$h=\{10^0,\ 10^{-1},\ 10^{-2},\ \ldots,\ 10^{-12}\}$分别计算其相对误差。当$x=1$时，运行

fidiff结果如下：

230

```
>> fidiff(1)
      h          fp         fpfd          Erel
   1.0e+00    2.71828    4.67077      7.18e-01
   1.0e-01    2.71828    2.85884      5.17e-02
   1.0e-02    2.71828    2.73192      5.02e-03
   1.0e-03    2.71828    2.71964      5.00e-04
   1.0e-04    2.71828    2.71842      5.00e-05
   1.0e-05    2.71828    2.71830      5.00e-06
   1.0e-06    2.71828    2.71828      5.00e-07
   1.0e-07    2.71828    2.71828      5.15e-08
   1.0e-08    2.71828    2.71828      2.43e-09
   1.0e-09    2.71828    2.71828      7.93e-08
   1.0e-10    2.71828    2.71828      5.69e-07
   1.0e-11    2.71828    2.71831      1.20e-05
   1.0e-12    2.71828    2.71871      1.59e-04
```

图5-7是相对误差对h的函数图。注意到上面的打印条目与图5-7中的横轴从右向左相对应。上面的数据和图示数据都表明，在$h \approx 10^{-8}$时，有限差分近似的误差最小。当$10^{-8}<h<1$时，相对误差的斜率为1，这和理论上所预测的截断误差与h之间为线性相关的结论相一致。当h小于10^{-8}时，由于舍入误差影响了$f(x+h)-f(x)$的计算，从而使相对误差增加。

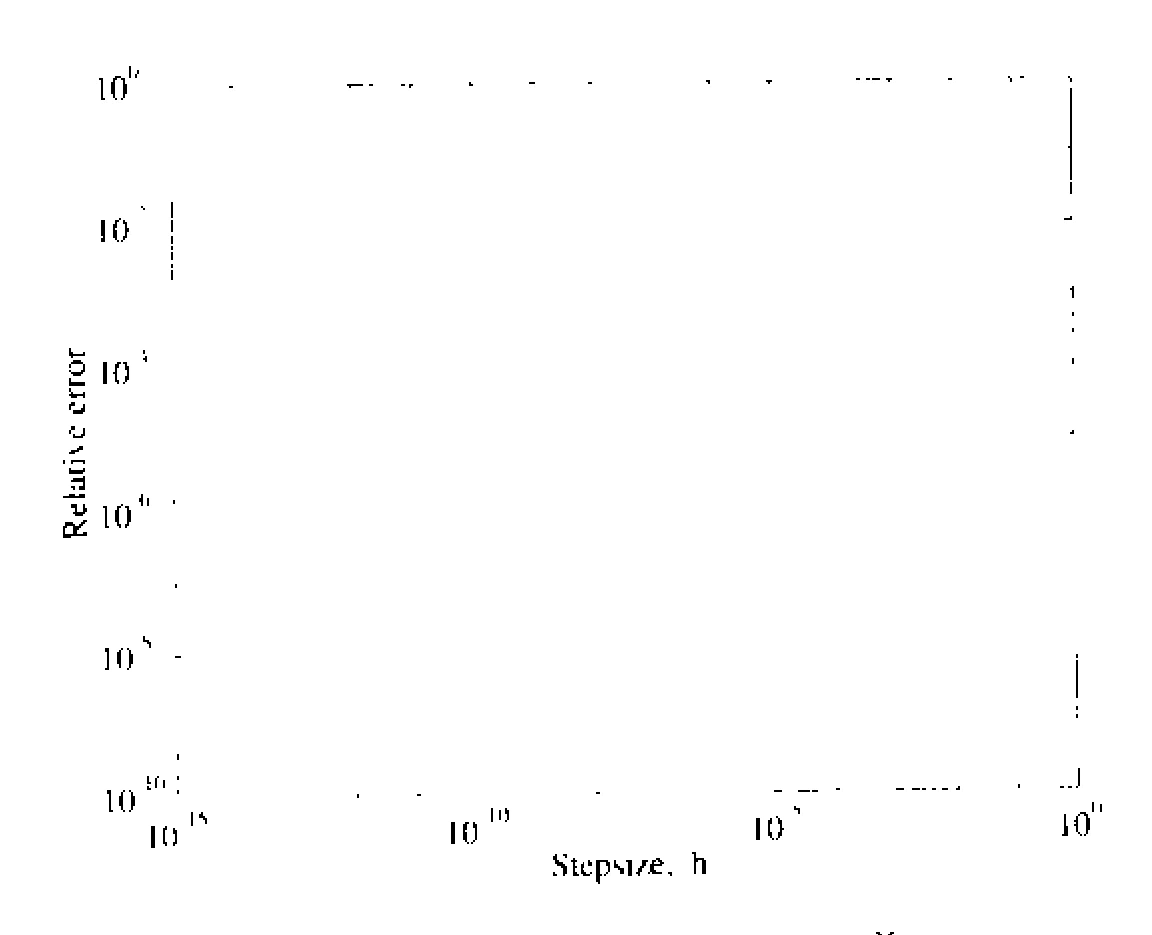


图5-7 $d(e^x)/dx$的有限差分的相对误差

231

本例得出的规律不仅适用于有限差分近似，还适用于其他许多数值计算。程序员或分析员能够控制截断误差的大小。本例中，通过减小h值这样更精确的技术，能够得到更小的截断误差。舍入误差会影响到截断误差的最小化。在许多计算中，能够避免舍入误差并使截断误差最小的h值范围很大。

5.4 小结

本章介绍了数值计算中固有的误差。要提高计算机内存的存储效率，就需要为每个数值

分配一个固定的二进制位数。这些固定位数的二进制数转换成十进制数会有上溢和下溢问题，这就限制了计算机能够存储的真实数据的范围。太大的数导致上溢错误，太小的数导致下溢错误。同时，限制每个数的位数也说明浮点数的精度是有限的。MATLAB中的数是按双精度数存储，因此，它们只有16个有效的十进制位。

尾数多于16个十进制位的实数和复数无法用双精度数精确地表示，所以，任意两个浮点数之间会存在间隙。浮点数x与距它最近的浮点数的距离是$|x|\varepsilon_m$，其中ε_m是机器精度。实际上，这意味着差值小于它们的平均值的ε_m倍的两个数在计算机中将作为同一个数存储。在双精度中，$\varepsilon_m \sim 2.2\times10^{-16}$。在MATLAB中，$\varepsilon_m$的值存储在内置变量eps中。

由于计算机中存储的数据位数有限，所以对浮点数的数学运算会带来舍入误差。在有许多步骤的数值运算（比如说数百万或数十亿步），舍入误差会逐渐破坏掉最后结果的最低有效位。舍入误差也可能导致灾难性的后果，即一步简单的操作也可能会导致很大的误差。当两个几乎相等的浮点数相减或两个相差好几个数量级的浮点数相加时，舍入误差最大。由于舍入误差，两个代数上等价（也就是说，在精确的算术计算中结果相同）的公式可能会出现不同的舍入特性。好的算法能够对公式进行代数变换从而找到使舍入误差最小的运算形式。例5.3示范了如何对熟悉的二次方程求根公式做这种变换。

232

在普通的计算任务中经常要比较两个浮点数的大小。用判断它们是否完全相等的方法是行不通的，在实际使用中通常是判定它们的差是否足够小。这一差值可以用绝对项（即差值的绝对值）或相对项（即差的绝对值除以一个合适的比例因子）表示。两个浮点数有意义的最小绝对差值是ε_m阶的，有意义的最小相对差值是ε_m乘以这两个浮点数的阶数。这些思想在趋近一个极限值的迭代计算过程中尤其重要（如何进行退出条件的判断）。

当用离散的代数方程或一系列的离散数据近似一个连续函数时，就会引起截断误差。截断误差和舍入误差之间的关系是相互独立的，互不影响的，但是它们都会影响到最后结果的精度。本章通过截断泰勒级数展开式介绍了截断误差。选择算法时一定要考虑到截断误差的影响，这种思想贯穿在本书后面的内容中。

表5-3列出了本章介绍舍入误差和截断误差用到的函数。这些程序都在NMM工具箱的errors目录下。

表5-3　介绍数值误差时用到的NMM工具箱中的函数

函　数	小　节	功能描述
Archimedes	5.3.2	圆内接n边形的周长。用于说明如何测量截断误差的阶数
demoTaylor	5.3.1	函数$f(x)=1/(1-x)$的泰勒级数近似的精度
epprox	5.2	在n的范围内计算$e = e^1 = \lim_{n\to\infty}(1+1/n)^n$，说明舍入误差可导致灾难性的后果
expSeriesPlot	5.3	计算函数exp (x)的级数并画出绝对误差对项数和的函数图
fidiff	5.3.2	计算dexp(x) / dx的一阶有限差分近似值。说明舍入误差和截断误差之间的折衷
halfDiff	5.1.3	将一个区间每次除以2直到结果接近零。证明了浮点数轴上两个数之间存在间隙
newtsqrt	5.2.4	用牛顿法计算某个数的开平方根。使用了不同的收敛准则
sinser	5.3	计算sine函数的级数表达式
testSqrt	5.2.4	在输入参数的某个范围内测试函数newtsqrt的正确性

233

补充读物

许多关于数值分析的介绍性读物都详细地说明了浮点运算的舍入行为。在对舍入误差的算法分析中，Burden和Faires[9]和Cheney和Kincaid[10]给出了很好的出发点，Stewart[68]和Gill等[28]则对浮点运算及其在实际计算中的影响给出了很好的概述。Bulirsch[70]和Isaacson 和Keller[40]对相同的资料给出了数学上的多种解决方法。Goldberg[29]从计算硬件的角度详细地介绍了浮点运算。Higham[36]综合地介绍了数值线性代数中的舍入误差分析。

习题

每个练习前圆括号中的数字表示练习的难度

1. （1）分别写出1、2、…、8对应的二进制数。
2. （1）分别写出5、21、35、64对应的二进制数。并用MATLAB内置函数dec2bin检验其结果的正确性。
3. *（1）把下列数转换成尾数为8位的规格化浮点数：0.4、0.5、1.5。
4. （3）按照算法5.1编写一个函数，用来计算浮点数尾数的位模式。将位模式存储在字符串向量中，余数r按标量存储，且能够为每一个k进行替换。思考：为什么内置函数dec2bin不能够用于这一算法？用上面的练习来检验函数的正确性。
5. （1）5.1.3节中通过计算realmax*10说明上溢。思考：为什么realmax+1不会出现上溢？
6. （3）为什么函数halfDiff中的循环上限设为1075？是否还存在输入值在该函数经过1075次循环后不能使δ=0呢（提示：参照6.3.1节的分析）？
7. （1）用5位数字运算手工求解例5.3中经典二次方程的两个根。再用6位数字运算计算一次，这两种情况下根的相对误差是多少？
8. （1）用4位数字运算采用手工办法分别求解例5.3中的公式（5-7）和（5-8）的两个根。两种情况下的相对误差是多少？
9. *（3）修改程序清单5-2中的函数epprox使它能在循环内保存向量n和abs(f-e)的值。（提示：创建向量nn和err，但不要改变循环索引n。）删除fprintf语句，将for语句改成for n = logspace(1, 16, 400)，以增加循环的步数。画出绝对误差对n的变化图，各点不要用线连接，而是用像“+”号这样的符号标出。解释n>10^7时的误差情况。 [234]
10. （1+）在图5-2中，与最大的浮点数realmax最近邻的一个双精度浮点数的近似值为多少？
11. （2+）利用5.2.1节中的代码编写一个m文件函数来计算ε_m的值。这段代码给出的ε_m与内置变量eps稍微有点不同。要求所编函数的返回值等于eps，要做哪些修改？并解释原因。
12. （2+）对习题11编写的m文件函数，用epsilon=epsilon/10替换epsilon = epsilon/2。修改后程序返回的ε_m值为多少？为什么要把除以2改成除以10？（提示：除以2和除以10是怎样影响位模式的？）
13. （2+）修改习题11所得函数，要求只有一个输入参数x_0。用x_0为参考值计算相对的ε_m

值。即，求出 $\hat{\varepsilon}_m$，对任意的 $\delta<\hat{\varepsilon}_m$，有$x_0+\delta=x_0$。对下面两个序列 x_0={1，10，100，1000}和x_0={1，2，4，8}分别计算 $\hat{\varepsilon}_m$。解释为什么上面两个序列值求得的 $\hat{\varepsilon}_m$ 按不同的比例增长？

14. （1）假设$c(x)=1-x^2/2!+x^4/4!$和$s(x)=x-x^3/3!+x^5/5!$分别近似于cos(x)和sin(x)。当x = 2、-1，0，1，2阶时，分别计算使用此公式的绝对误差和相对误差？
15. （2+）对x = `logspace(-12, 2, 100)`，用公式sinh(x)=（e^x-e^{-x}）/2来计算sinh(x)。画出使用此公式代替内置函数`sinh`（假设给定sinh(x)值是正确的）带来的绝对误差和相对误差的图形。当x很小时误差是如何产生的？
16. （2+）使内置函数`linspace`生成一个均匀步长的n维向量，语句如下所示：

```
d1+(0:n-2)*(d2-d1)/(n-1) d2
```

用语言描述上式等号右边的表达式的含义。为什么不能写成下列形式：

```
d1+(0:n-1)*(d2-d1)/(n-1)
```

（提示：在命令提示符后输入`type linspace`得到该内置函数的详细清单。）

17. （3　）程序清单5-10中的函数`linsp1`、`linsp2`、`linsp3`都是内置函数`linspace`的各种变种形式。通过实验方法比较这些函数和`lintest`函数：

235

程序清单5-10　函数`linspace`的几个效率较低变种形式

```
function x = linsp1(x1,x2,n)
% linsp1  Generate a vector of equally spaced values, version 1.
%
% Synopsis:  x = linsp1(x1,x2)
%            x = linsp1(x1,x2,n)
%
% Input:     x1, x2  = lower and upper limits of vector elements
%            n = (optional) number of elements to generate. Default: n=100
%
% Output:    x = vector of n equally spaced values  in x1 <= x <= x2
if nargin<3,  n=100;  end
dx = (x2-x1)/(n-1);
x(1) = x1;
for k=2:n
  x(k) = x(k-1) + dx;
end

function x = linsp2(x1,x2,n)
% linsp2  Generate a vector of equally spaced values, version 2.
%
% Synopsis:  x = linsp2(x1,x2)
%            x = linsp2(x1,x2,n)
%
% Input:     x1, x2  = lower and upper limits of vector elements
%            n = (optional) number of elements to generate. Default: n=100
%
% Output:    x = vector of n equally spaced values in x1 <= x <= x2
if nargin<3,  n=100;  end
dx = (x2-x1)/(n-1);
x(1) = x1;
for k=2:n
  x(k) = x1 + (k-1)*dx;
end
```

```
function x = linsp3(x1,x2,n)
% linsp2  Generate a vector of equally spaced values, version 3.
%
% Synopsis:  x = linsp3(x1,x2)
%            x = linsp3(x1,x2,n)
%
% Input.     x1, x2  = lower and upper limits of vector elements
%            n = (optional) number of elements to generate. Default: n=100
%
% Output:    x = vector of n equally spaced values  in x1 <= x <= x2
if nargin<3,  n=100;  end
dx = (x2-x1)/(n-1);
x = x1:dx:x2;
```

236

```
function lintest
% lintest  Compare schemes for generating a vector of equally spaced values

fprintf('  n      norm(y1-y)     norm(y2-y)     norm(y3-y)\n');
for n=[4 5 6 9 10 20 50 100]
  y  = linspace(0,1,n);
  y1 = linsp1(0,1,n);      y2 = linsp2(0,1,n);      y3 = linsp3(0,1,n);
  e1 = norm(y1-y,'inf');  e2 = norm(y2-y,'inf');  e3 = norm(y3-y,'inf');
  fprintf('%4d  %12g  %12g  %12g\n',n,e1,e2,e3);
end
```

运行函数lintest结果如下:

```
>> lintest
     n    norm(y1-y)     norm(y2-y)    norm(y3-y)
     4            0              0    1.11022e-16
     5            0              0              0
     6  1.11022e-16    1.11022e-16              0
     9            0              0              0
    10  2.22045e-16    1.11022e-16    1.11022e-16
    20  4.44089e-16    1.11022e-16    1.11022e-16
    50  7.77156e-16    1.11022e-16    1.11022e-16
   100   1.9984e-15    1.11022e-16    1.11022e-16
```

显然、函数linsp1、linsp2和linsp3与linspace的结果不同。为什么$n>10$时$\|y1-y\|_\infty$会增大？为什么对所有测试数据n，$\|y2-y\|_\infty$和$\|y3-y\|_\infty$都不大于$\varepsilon_m/2$？可以参照7.1.2节末关于向量范数的背景介绍。

18. （2）对程序清单5-3中的函数newtsqrt作必要的修改重新生成例5.6中的结果。最好是分别编写newtsqrta和newtsqrtr两个函数来实现绝对误差准则和相对误差准则。作为程序的诊断信息、对每个newtsqrt函数的输入参数打印出收敛所需要的迭代次数。

19. *（2+）实现程序清单 5-3中函数newtsqrt的组合公差$|x_k - x_{k-1}| < \max\{\Delta_a, \delta_r|x_{k-1}|\}$。注意，要求$\Delta_a$和$\delta_r$是newtsqrt函数的两个不同的输入。作为程序的诊断信息，对每个newtsqrt函数的输入参数打印出收敛所需要的迭代次数。根据例5.6的结果、Δ_a和δ_r的默认值最好是多少？如果$\Delta_a=\delta_r$、收敛情况会与例5.6的结果相差很大吗？使用绝对公差对算法有作用吗？

20. （2+）公式（5-12）中的收敛准则写成…$<\Delta_a$和…$<\delta_r$的形式、但是例5.6中的收敛准则却写成

```
while abs(r-rold)/rold > delta & it<maxit
```

237

这种形式。这有错误吗？while语句中是否要用<号来代替>号？为什么？

21.（2+）使用程序清单5-5中的`sinser`函数和收敛公差的固定值`tol = 5e-9`，来观察其收敛速度对sin(x)中的x的敏感性。特别地，画出达到收敛的n值以及$0 \leqslant x \leqslant 8\pi$中的10个$x$点的对应图。你可能要修改函数`sinser`使它返回$n$值，而且还需要确定级数中最少50项来观察$n$对$x$的变化趋势。

22.（2+）修改程序清单5-5中的函数`sinser`，使它满足的收敛准则为

$$\frac{T_k}{T_1} < \delta$$

其中T_k是当前与S_k相加的那一项。这和原始的收敛准则差别大吗？对相同的δ值，比较这一准则和原始准则下收敛所需要的项数。你认为哪种收敛准则更好？

23.*（2+）用程序清单5-5中的`sinser`函数分别计算$x = \pi/6$、$5\pi/6$、$11\pi/6$和$17\pi/6$时所对应的sin(x)值，根据sine函数的周期性来修改函数`sinser`，以避免前面输入的x序列所显示出来的不受欢迎的情况（提示：`for...end`循环里的语句不要做任何修改）。

24.（2+）参照程序清单5-5中的函数`sinser`，编写函数计算ln(1+x)的级数表达式。使用所编函数分别计算x= 0.01、0.1、0.9、1.0时ln(1+x)的值。

25.（3+）访问网页http://www.netlib.org/，并浏览fn目录。下载计算sin(x)或cos(x)的程序，阅读源代码并写一个简单的报告，描述这些代码如何计算sin(x)或cos(x)值。用了多少Fortran函数？

26.（3+）用下列极限得到欧拉常量γ

$$\gamma = \lim_{n\to\infty}\gamma_n, \quad 其中\ \gamma_n = \left[1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}-\ln n\right]$$

γ一般出现在特殊数学函数的分析表达式中，如指数积分和一些贝塞尔（Bessel）函数中。由于$n\to\infty$时$\gamma_n\to$常量，因此可以用$\gamma_n - \gamma_{n-1}$作为收敛公差。根据γ_n的定义，写出$\gamma_n - \gamma_{n-1}$对n的函数关系，并用这一表达式估算出使$\gamma_n - \gamma_{n-1} < 5\times10^{-9}$所需要的项数。注意，利用$\gamma_n - \gamma_{n-1}$不可能求得$n$的值，只能用图形来估算，编写m文件函数计算$\gamma$值。当用$\gamma_n - \gamma_{n-1}$作为收敛准则时，能够准确地估算截断误差吗？

27.*（1）推导例5.9中的泰勒级数展开式$P_1(x)$、$P_2(x)$、$P_3(x)$。

238
|
239

28.（2）利用程序清单5-7中的`demoTaylor`函数为模型，在x_0=1的邻域内求出对$f(x)$ = ln(x)的泰勒级数展开式$P_1(x)$、$P_2(x)$、$P_3(x)$并画图。

29.（1+）利用sin(x)的级数展开式说明例5.11中的阿基米德算法的截断误差为$E(d, n) = O(1/n^2)$。

30.（3+）修改程序清单5-9函数`fidiff`中的`logspace`语句，在区间$10^{-12} \leqslant h \leqslant 1$上创建一个有400个对数分布点的向量。当$x = 1$时运行修改后的函数，并解释当$h<10^{-7}$的误差。将`fprintf`语句注释掉或者把它放入`if...end`结构中也会有助于解题。

第6章 一元方程 $f(x)=0$ 求根

在许多实际应用中，方程不能直接处理或通过分析得到显式的解；即使可以通过分析求出解，但过程也很繁琐费时。这类问题重要的一个方面就是求满足方程 $f(x)=0$ 的 x 值，这些 x 值称为方程的零点或根。所有含一个自变量的函数都能够写成这种形式。图6-1列出了本章将要介绍的主要内容，第8章进一步介绍了求解非线性方程组的方法。

本章主题

1 **预备知识**

本节将介绍数值求根方法中常遇到的共性问题。当根 x 落在 $x_{left} < x < x_{right}$ 中时，称数据对（x_{left}，x_{right}）为**有根区间**（bracket）。这里还将介绍寻找有根区间并带画图功能的一个MATLAB程序。

2 **定点迭代**

定点迭代是指首先猜测一个根并用它作为输入对公式进行计算，然后返回修正后的根作为输出，其成功与否取决于迭代公式的选取。

3. **二分法**

给定初始有根区间，并在这个有根区间内有步骤地进行等分的方法称为**二分法**（bisection method）。这种方法虽然收敛速度比较慢，但是在任何情况下，它都会收敛。

4. **牛顿法**

牛顿法（Newton's Method）首先需要方程的初始猜测根和 $f(x)$ 的斜率，然后用外推法预测 $f(x)$ 在何处经过 x 轴。当迭代收敛时，用这种方法求解方程根的速度很快，但是如果在迭代过程中出现 $f'(x)=0$，那么它就会发散。

5. **割线法**

割线法（secant method）不用分析方法来求 $f'(x)$，而是通过前两次猜测的 x 值处的 $f(x)$ 来近似 $f'(x)$。割线法和牛顿法的收敛速度几乎一样快，但同样地，当 $f'(x)=0$ 时也会发散。

6. **混合法**

混合法综合了几种不同的求根法的特性，它能够快速可靠地收敛并得到解。内置函数fzero是采用混合法求根的常用程序。

7 **多项式的根**

用数值方法求解多项式的根有很大困难。函数roots通过求解**特征值问题**（eigenvalue problem）来得到多项式的根。本节将介绍特征值的构造和roots函数的使用方法。

图6-1 第6章主题

例6.1 设计一个野餐桌的桌腿

思考图6-2中的野餐桌如何设计，图6-3中标出了桌腿所对应的尺寸。桌腿架的宽度 w 和高度 h 决定了桌面的规格，而桌腿所选用的型材决定了 b。为了美观实用，先选取桌腿的 h、w 和 b 值，然后再计算其他尺寸。而且，制造桌腿时，要求 a、c、d_1 和 d_2 是已知的。

240
241

根据图6-3中的几何结构得：

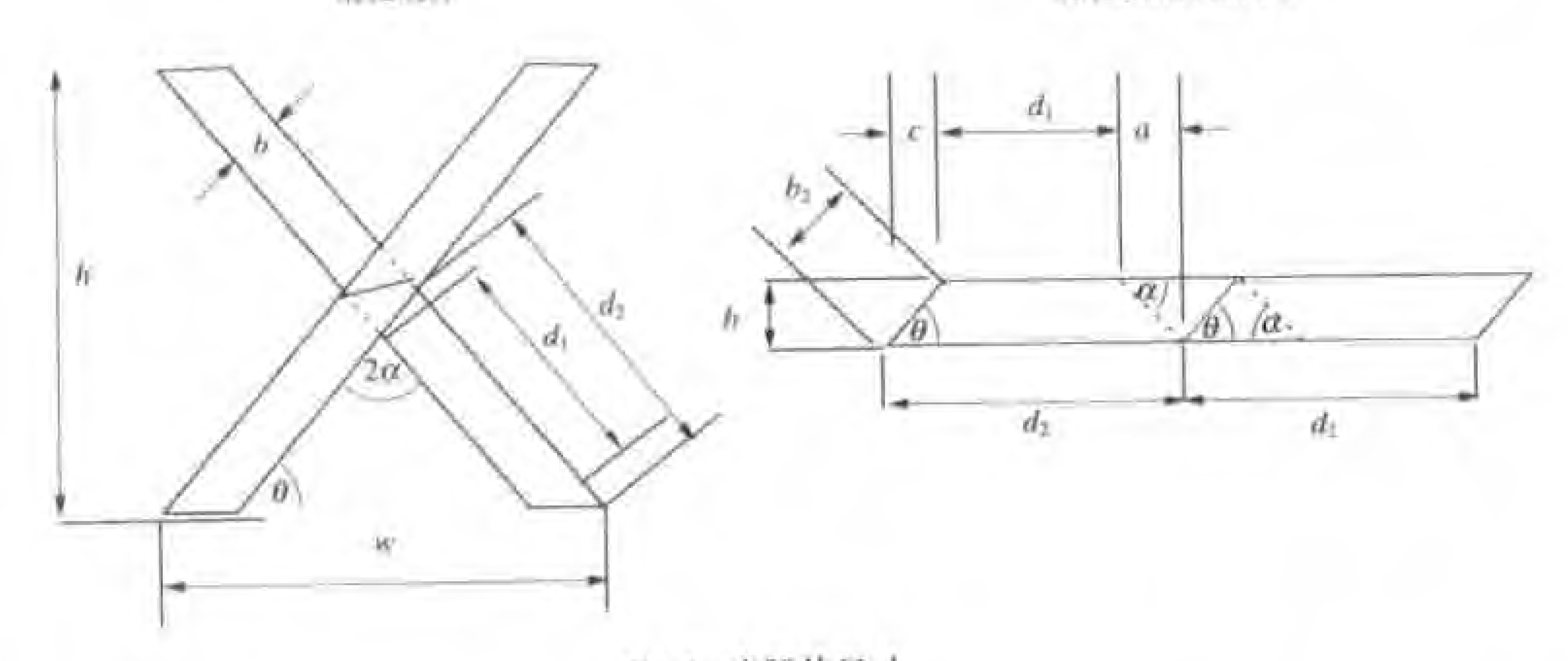
$$h=2d_2\sin\theta,\quad w=2d_2\cos\theta+b_2,\quad b=b_2\sin\theta \tag{6-1}$$

合并前面的方程并消去 d_2，得

$$w\sin\theta=h\cos\theta+b \tag{6-2}$$



图6-2 一个野餐桌，当它的桌腿型材宽度以及桌子的总体高度和宽度确定后，桌腿的其他尺寸就可以用求根程序计算出来



242

图6-3 桌腿的尺寸

解方程(6-2)求出θ，可以通过方程（6-1）的第一个式子得到d_2。若$b = 0$，那么很明显有$h/w = \tan\theta$。另外的主要尺寸d_1的值可以通过下列式子计算出来：

$$d_1 = d_2 - a - c$$

其中

$$a = \frac{b}{\tan\alpha}, \quad c = \frac{b}{\tan\theta}, \quad \alpha = \frac{\pi}{2} - \theta$$

虽然不明显，但是方程（6-2）确实存在解析解（参照问题1）。对这种方程，我们要么就花时间来求出其解析解，要么就用本章介绍的数值方法由已知的w、b和h求出θ值，然后再计算其他值。

6.1 预备知识

本节是求根法的一个序曲，介绍了选择求根方法的折衷方案并给出了求根法数值处理过程的总览。本节也介绍了寻找根的初始猜测值的范围划分法。

6.1.1 总则

求根方法的选取在一定程度上和遇到的具体问题有关。下列是一些需要考虑的问题：

- 这是要经常求解的特殊方程吗？若是，那么最好使用理论分析或代数变换来求解。这样能用很短的代码设计出定制的迭代方案来（参照文献[2]）。
- 精度要求是多少？工程计算中只需要很少的有效位，一个简单的求根程序就足够了。
- 数值方法的速度要多快？是否够健壮？如果求根程序被嵌入到另一个程序中，而且这个程序不断地自动改变$f(x)$的参数，这时，就需要求根程序很健壮。健壮的程序不受初始猜测值的影响，并且它的收敛情况不受$f(x)$在参数空间中行为的影响。
- 函数是多项式吗？这时通常不用常规方法，而用二分法、牛顿法或割线法这样的特殊程序来求解。
- 函数存在奇点吗？有些求解程序会收敛到奇点和根上，我们应该尽量避免收敛到奇点的情况。 243

6.1.2 基本的求根程序

图6-4画出了某些在x规定范围内有两个根x_1和x_2的函数$f(x)$。从图形上能够很明显地看出方程的根，这也是求解方程$f(x)$的根的一个方法。除初始猜测根以外，画图会暴露出某些函数的一些病态特征，这有可能会导致求根程序最终失败。

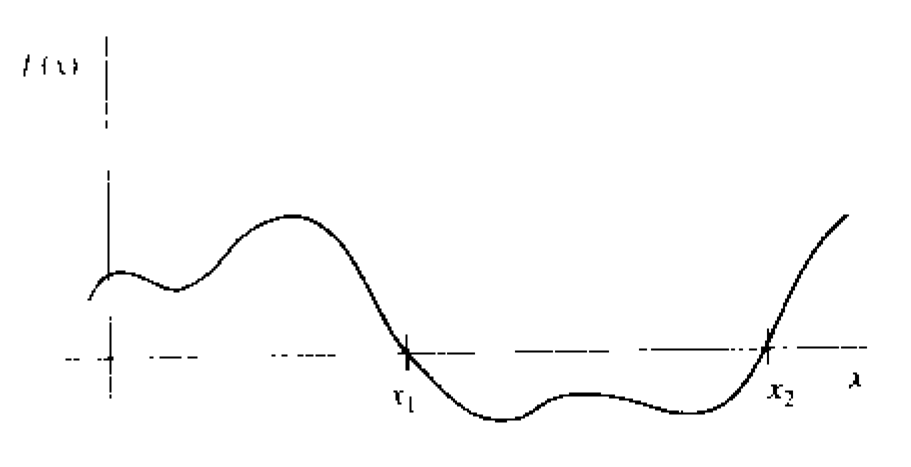


图6-4　一般标量函数$f(x)=0$的根

观察画出的$f(x)$图形获得根的初始猜测值后，就可使用数值求根程序来细化初始的猜测根。如果不出意外的话，结果将是一个逐渐接近实际根的序列。假设x_k是第k次迭代的近似根，那么如果极限$\lim_{k\to\infty} x_k=\xi$存在，迭代方式就会收敛，其中$\xi$为方程的根。在实际求根应用中，我们希望收敛的迭代次数越少越好，所使用的自动求根算法也要能够判断出在何时停止迭代。

用自动求根程序求到方程的根后，最好把x值代入$f(x)$中看$f(x)\approx 0$是否成立。这样既能检测所求根是否是奇点，还可以指出所求根满足方程的情况。 244

6.1.3 根区间划分

区间划分法是在x轴上一个比较大的区间中粗略求根的过程，结果是得到一组几乎肯定包含根在内的子区间。对子区间递归使用区间划分法可以得到更准确的解，但是这样做效率低下，不推荐使用。区间划分法解决了“根在哪里？”的问题，后面几节解决了“在给定初始

猜测根的情况下，求根值”的问题。

区间划分法首先把给定的大区间划分成若干小的子区间，然后在每个子区间的两端检查函数符号，看函数是否通过x轴，这是方程在这个子区间中是否有解的标志。如果$f(a)$和$f(b)$符号不同，方程就在区间$[a, b]$上有解。图6-5的左图说明函数在$x = a$和$x = b$之间通过x轴。图6-5的右图说明符号的变化也可能是由于$f(x)$出现了奇点。

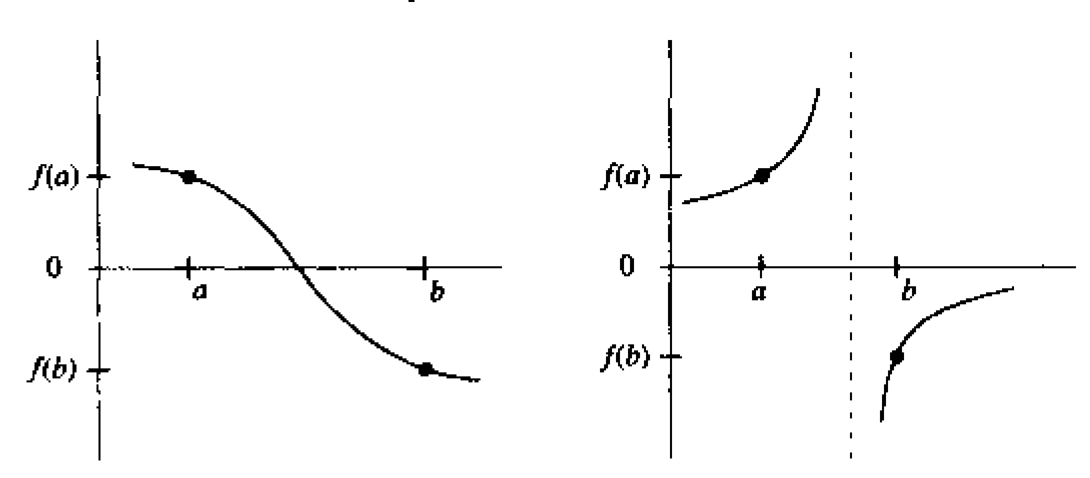


图6-5 依照$f(x)$的符号改变来测试有根区间会把根和奇点一起包括进去

区间划分法的算法描述如下所示：

算法6.1 区间划分法

given: $f(x)$, x_{min}, x_{max}, n
$dx=(x_{max}-x_{min})/n$ (设置根区间大小)
$a=x_{min}$ (初始化测试根区间的左端点)
$i=0$ (初始化计数器)
while $i<n$
  $i \leftarrow i+1$
  $b=a+dx$
if $f(x)$ change sign in $[a,b]$
  save$[a,b]$for further root-finding
 end
 $a=b$
end

245

测试部分“if $f(x)$ changes sign in $[a,b]$”还需要更详细的实现，下面是一个简单的实现方法

$$f(a) \times f(b) < 0 ?$$

这种方法虽然很好，但是在浮点运算中会存在问题，如果$f(a)$和$f(b)$的值都很小（即当区间的两端都很接近某个根时），上面两个数的乘积可能会导致下溢从而导致对符号变化的检测出错。例如：

```
>> format long e
>> fa = 1e-120;  fb = -2e-300;
>> fa*fb
ans =
     0
```

因为`fa*fb < realmin`，结果下溢，被置为零值（参照5.1.3节）。这种含糊的结果会引起很难发现的逻辑错误。使用MATLAB中的`sign`函数进行判别比较可靠，即：

```
>> fa = 1e-120;  fb = -2e-300;
>> sign(fa)~=sign(fb)
ans =
     1
```

不管fa和fb的值多小，当它们的符号不同时，表达式sign(fa) =sign(fb)总会返回真实的值（参照练习30）。

程序清单6-1中的函数brackPlot会自动寻找根的区间范围并返回该区间值，另外，它还会画出函数图并用灰色的框把所寻得的区间框起来。如函数对照表中提到的，brackPlot函数有两种调用形式：

```
Xb = brackPlot(fun,xmin,xmax)
Xb = brackPlot(fun,xmin,xmax,nx)
```

其中第一个输入变量fun是一个字符串变量，它包含了计算$f(x)$的MATLAB函数的名称，它可以是内置函数，但一般来说它是一个用户自定义函数（参照例6.3）。参数xmin和xmax是测试区间的最左端和最右端。第四个参数nx是可选参数，表示划分区间的个数，系统默认值是20。输出Xb是由有根区间端点构成列数为两列的矩阵，它的每一行是区间的两个端点。

246

在典型的求根程序中，使用根区间划分算法得到的近似根还比较粗略。下一步要使近似根更加细化，这将在后面几节介绍。

程序清单6-1 使用MATLAB函数brackPlot得到有根区间向量以便其他求根程序使用

```
function Xb = brackPlot(fun,xmin,xmax,nx)
% brackPlot  Find subintervals on x that contain sign changes of f(x)
%
% Synopsis:  Xb = brackPlot(fun,xmin,xmax)
%            Xb = brackPlot(fun,xmin,xmax,nx)
%
% Input:     fun = (string) name of mfile function that evaluates f(x)
%            xmin,xmax = endpoints of interval to subdivide into brackets.
%            nx = (optional) number of samples along x axis used to test for
%                 brackets.  The interval xmin <= x <= xmax is divided into
%                 nx-1 subintervals.  Default:  nx = 20.
%
% Output:    Xb = two column matrix of bracket limits.  Xb(k,1) is the left
%                 (lower x value) bracket and Xb(k,2) is the right bracket for
%                 the k^th potential root.  If no brackets are found, Xb = [].

if nargin<4, nx=20; end

% --- Plot f(x) on interval xmin <= x <= xmax
xp = linspace(xmin,xmax);  yp = feval(fun,xp);
plot(xp,yp,[floor(xmin) ceil(xmax)],[0 0]);
grid on;  xlabel('x');  ylabel(['f(x) defined in ',fun,'.m']);

% --- Save data used to draw boxes that indicate brackets
ytop = max(yp);  ybot = min(yp);            %  y coordinates of the box
ybox = 0.05*[ybot ytop ytop ybot ybot];     %  around a bracket
c = [0.7 0.7 0.7];                          %  RGB color used to fill the box

% --- Begin search for brackets
x = linspace(xmin,xmax,nx);    %  Vector of potential bracket limits
f = feval(fun,x);              %  Vector of f(x) values at potential brackets
```

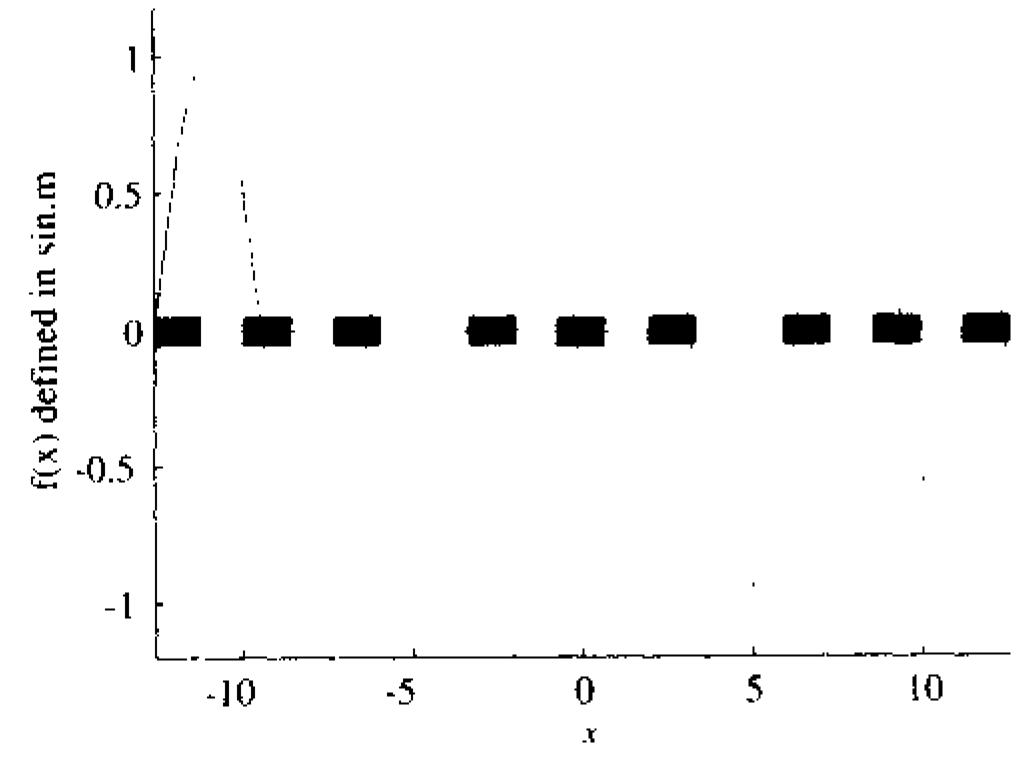
```
nb = 0;  Xb = [];                  %  Xb is null unless brackets are found
for k = 1:length(f)-1
  if sign(f(k))~=sign(f(k+1))  %  True if sign of f(x) changes in the interval
    nb = nb + 1;
    Xb(nb,1) = x(k);            %  Save left and right ends of the bracket
    Xb(nb,2) = x(k+1);
    hold on;  fill([x(k) x(k) x(k+1) x(k+1) x(k)],ybox,c); %  Add filled box
  end
end
hold off
if isempty(Xb)                  %  Free advice
  warning('No brackets found.  Check [xmin,xmax] or increase nx');
end
```

[247]

例6.2 用根区间划分法求sin(x)的根

sin(x)根的值是π的整数倍，用它能够方便地测试brackPlot函数。使用函数brackPlot在$-4\pi < x < 4\pi$区间上计算$\sin(x) = 0$，得到图6-6中的图形和下面的文本输出：

```
>> brackPlot('sin',-4*pi,4*pi)
ans =
  -12.5664  -11.2436
   -9.9208   -8.5980
   -7.2753   -5.9525
   -3.3069   -1.9842
   -0.6614    0.6614
    1.9842    3.3069
    5.9525    7.2753
    8.5980    9.9208
   11.2436   12.5664
```



[248]

图6-6 用函数brackPlot得到的$\sin(x) = 0$在$-4\pi < x < 4\pi$上的有根区间

注意！根并不是在有根区间的中心，根区间划分算法只能保证函数在有根区间中改变符号，它不能说明根与区间左端或右端具体相差多远

例6.3 用根区间划分法求用户自定义函数f(x)的根

任何能够用MATLAB代码表示的函数$f(x)$都能用函数brackPlot求根。通常，用简单的

MATLAB语句就能够求出$f(x)$的根，但是这并不必要。计算$y=f(x)$的m文件函数要输入一个向量x并返回向量y，因此用MATLAB表达式计算$f(x)$的值通常要用到数组操作符（.*、./、.^）

考虑下列函数

$$f(x)=x-x^{1/3}-2=0 \tag{6-3}$$

它存在一个在$x=3.5$附近的单实数根，程序清单6-2中的函数fx3实现了对方程（6-3）的计算，其中语句x.^(1/3)中的数组操作符使函数既能对向量操作，也能对标量操作。利用函数brackPlot得到方程（6-3）的有根区间并画出了图6-7所示图形：

```
>> brackPlot('fx3',0,5)
ans =
    3.4000    3.6000
```

程序清单6-2 用MATLAB函数fx3计算方程(6-3)的左端

```
function f = fx3(x)
% fx3   Evaluates f(x) = x - x^(1/3) - 2
f = x - x.^(1/3) - 2;
```

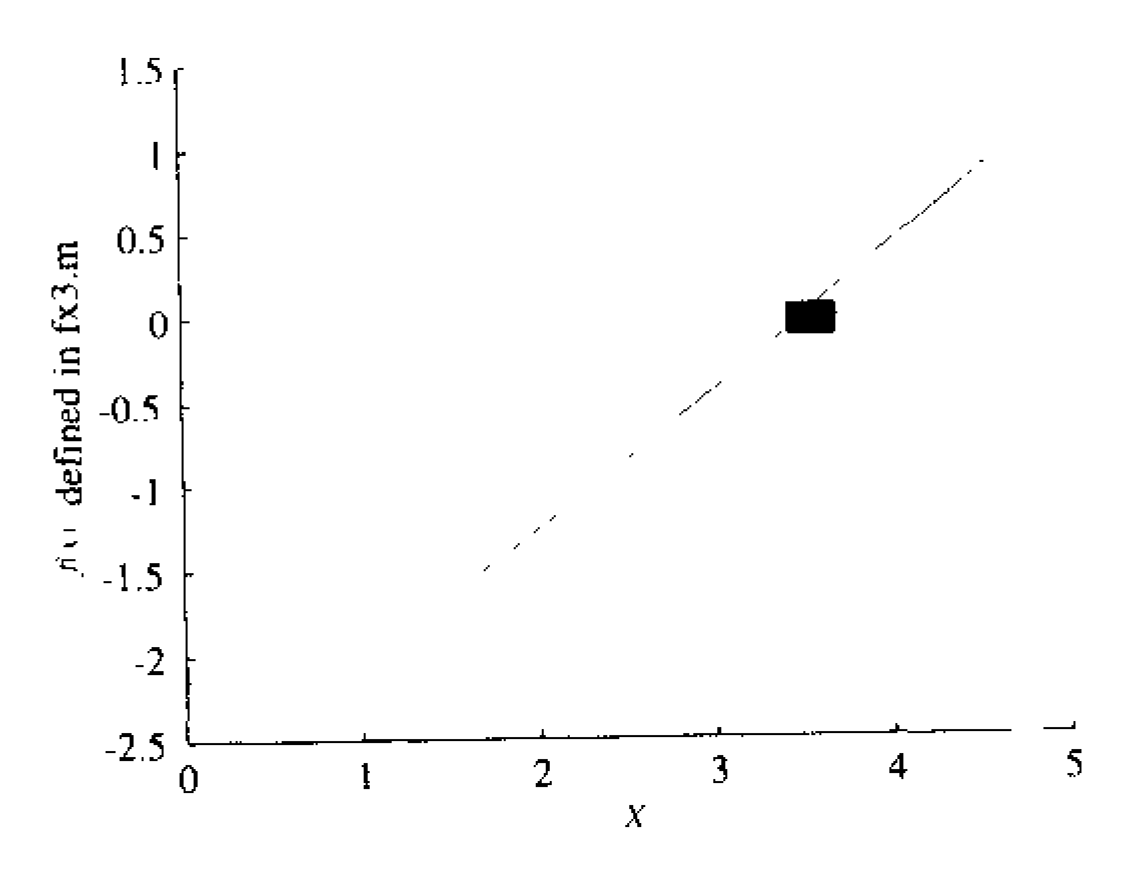


图6-7 函数$f(x)=x-x^{1/3}-2$的图形，它的根在$x=3.5$附近

249

6.2 定点迭代

定点迭代（*fixed-point iteration*）包括简单求根程序和求根算法分析的理论框架。作为简单的求根程序，定点迭代常用于手工计算中。要解$f(x)=0$，首先将某个包含x的项分离，方程变成下列形式

$$x_{new}=g(x_{old})$$

其中，$g(x)$称为迭代函数（*iteration function*）。给定$g(x)$的公式，下列算法用定点迭代法求$f(x)=0$的根：

算法6.2 定点迭代

```
initialize: x0=...
for k = 1,2,...
    xk = g(xk-1)
    if converged, stop
end
```

作者有意不把步骤“if converged, stop”写得很明确，因为其中包含了收敛准则的内容，这将在6.3.2节中加以介绍。在对x迭代的过程中只需要保存两个变量，这里不需要产生x一系列值的向量，而可能只使用两个变量，如x_{new}和x_{old}，每次迭代完成后用x_{new}值替换x_{old}的值即可。

定点迭代算法的名称源于它在收敛中表现出的特性：循环地将输出作为输入，而不改变输出值。假设由迭代函数$x_k = g(x_{k-1})$产生的序列x_1、x_2，...在ξ处收敛(即$\lim_{k\to\infty} x_k = \xi$)，我们称$\xi$为$g(x)$的定点（*fixed point*）。为了求$f(x)=0$的根，可能需要构造许多收敛（或发散）特性不同的$g(x)$。其中，函数$g(x)$的选取对定点迭代成功与否起关键作用，下面两个例题就说明了这一点。 250

例6.4 $x - x^{1/3} - 2=0$的定点迭代

如例6.3所述，函数

$$f(x) = x - x^{1/3} - 2 = 0$$

的根在$x=3.5$附近。对方程$f(x)$中x的第一项或第二项分离可得到两个定点迭代函数，如下所示：

$$g_1(x) = x^{1/3} + 2，\text{和}\ g_2(x) = (x-2)^3$$

这里还存在一个迭代函数

$$g_3(x) = \frac{6+2x^{1/3}}{3-x^{2/3}}$$

但$g_3(x)$不是对$f(x)$进行代数变换得到的。下表列出了当初始猜测值为$x_0=3$时，不同的$g(x)$函数的迭代结果：

k	$g_1(x_{k-1})$	$g_2(x_{k-1})$	$g_3(x_{k-1})$
0	3	3	3
1	3.4422495703	1	3.5266442931
2	3.5098974493	−1	3.5213801474
3	3.5197243050	−27	3.5213797068
4	3.5211412691	−24389	3.5213797068
5	3.5213453678	-1.451×10^{13}	3.5213797068
6	3.5213747615	-3.055×10^{39}	3.5213797068
7	3.5213789946	-2.852×10^{118}	3.5213797068
8	3.5213796042	∞	3.5213797068
9	3.5213796920	∞	3.5213797068

$g_1(x)$迭代函数收敛，但是$g_2(x)$迭代函数是发散的，即使猜测值为$x=3.5213797$，$g_2(x)$仍

然会发散。$g_3(x)$只迭代3次就能够收敛，且有效位数为11位，相比之下，$g_1(x)$经过9次迭代后才能保证解的前7位有效位正确。

例6.5 $x-12x^{1/3}+12=0$的定点迭代

方程

$$f(x)=x-12x^{1/3}+12 \tag{6-4}$$

和方程（6-3）的形式完全相同，只是系数不同，同样使用定点迭代法求根。图6-8显示方程(6-4)有两个根，一个在$x=1.5$附近，另一个在$x=21$附近，它的三个迭代函数如下所示：

$$g_1(x)=12x^{1/3}-12 \quad g_2(x)=\left(\frac{x+12}{12}\right)^3 \quad g_3(x)=\frac{8x^{1/3}-12}{1-4x^{-2/3}}$$

251

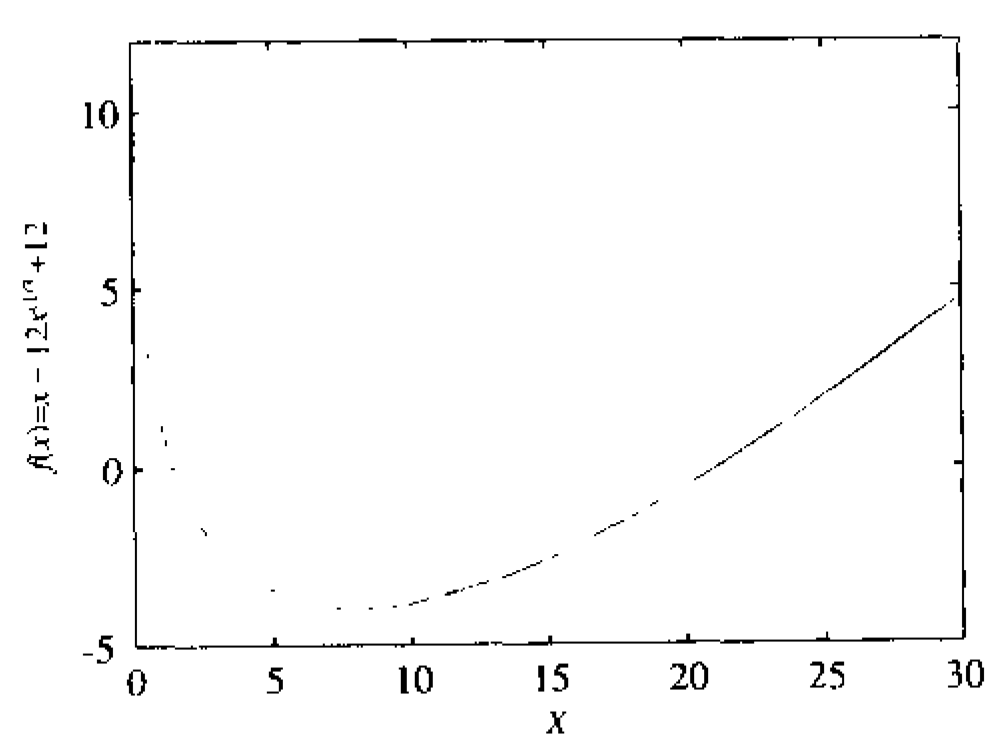


图6-8 $f(x)=x-12x^{1/3}+12$的函数图，它的两个根分别在$x=1.5$和$x=21$附近

给定合理的初始猜测值，上面三个公式都会收敛。例如，假设初始猜测值为$x=1$，得到如下迭代结果：

n	x_1	x_2	x_3
0	1.0000000000	1.0000000000	1.0000000000
1	0.0000000000	1.2714120370	1.3333333333
2	12.0000000000	1.3527192196	1.3879068816
⋮	⋮	⋮	⋮
9	21.4098558464	1.3889825412	1.3889927600
10	21.3236311792	1.3889895796	1.3889927600

给定初始猜测值$x=1$时，$g_1(x)$收敛于$x=21$附近，而$g_2(x)$和$g_3(x)$都收敛于$x=1.3899....$如果初始猜测值为$x=20$，那么迭代就不同了，其过程如下所示：

n	x_1	x_2	x_3
0	20.0000000000	20.0000000000	20.0000000000
1	20.5730113991	18.9629629630	21.2535347737
2	20.8811653906	17.1784431386	21.2248257659
⋮	⋮	⋮	⋮

9	21.2211516518	1.4826466151	21.2248116633
10	21.2229017888	1.4183444605	21.2248116633

252

此时，$g_1(x)$和$g_3(x)$都收敛于根$x=21.2248\cdots$，而$g_2(x)$收敛于根$x=1.3899\cdots$。实际上，对于任意正数猜测值，$g_1(x)$都会收敛于$x=21.2248$，而$g_2(x)$也都收敛于$x=1.3899$。

定点迭代的收敛性

通过对定点迭代（参见文献[9、53、68]）的分析，我们知道，如果在区间$a \leqslant x \leqslant b$中有根，且满足

$$\text{对所有}x:\ a \leqslant x \leqslant b,\ \text{有}\ |g'(x)|<1\ \text{且}a \leqslant g(x) \leqslant b$$

时，迭代在此区间上收敛。若$-1<g'(x)<0$，则迭代在收敛的过程中会发生振荡。若$0<g'(x)<1$，迭代单调收敛。练习7就是要验证收敛准则和前面例题得到的结果一致。

6.3 二分法

二分法可简单地表达为：给定一个有根区间，只要此区间包含根，就反复地将这一区间进行平分。由于每次都将区间分成两半，故而二分法有时也称为区间平分法。只要原始区间包含所求根，二分法总是收敛的。给定起始区间$[a,b]$，对方程根进一步的近似值就是区间的中点

$$x_m=(a+b)/2$$

上面式子可以变换成不易受舍入误差影响的形式，如下所示：

$$x_m=a+(b-a)/2$$

求出中点后，算法将测试函数在区间中点和左端或右端之间是否改变符号。然后将根据测试结果用中点替换区间的一个端点，以排除掉不含根的那个半区间。二分法的思想包含在下列算法中：

253

算法6.3 二分法

```
initialize: a = ...,b = ...
for k = 1,2,...
    x_m = a+(b − a)/2
    if sign(f(x_m)) = sign(f(x_a))
        a = x_m
    else
        b = x_m
    end
    if converged, stop
end
```

图6-9描述了此程序应用于一个典型的$f(x)$函数求根的过程。算法的起始区间为x轴上的$[a,b]$区间。第一个根近似值为x_1，它是a和b的中点。根不在x_1和b之间（函数$f(x)$在x_1和b处的符号相同），因此，用x_1替换b，并把x_1作为区间的右端点。第二个根近似值为x_2，它是a和x_1的

平均值。当根所在区间足够小或对应于区间中点处的函数值足够小时，程序终止运行。

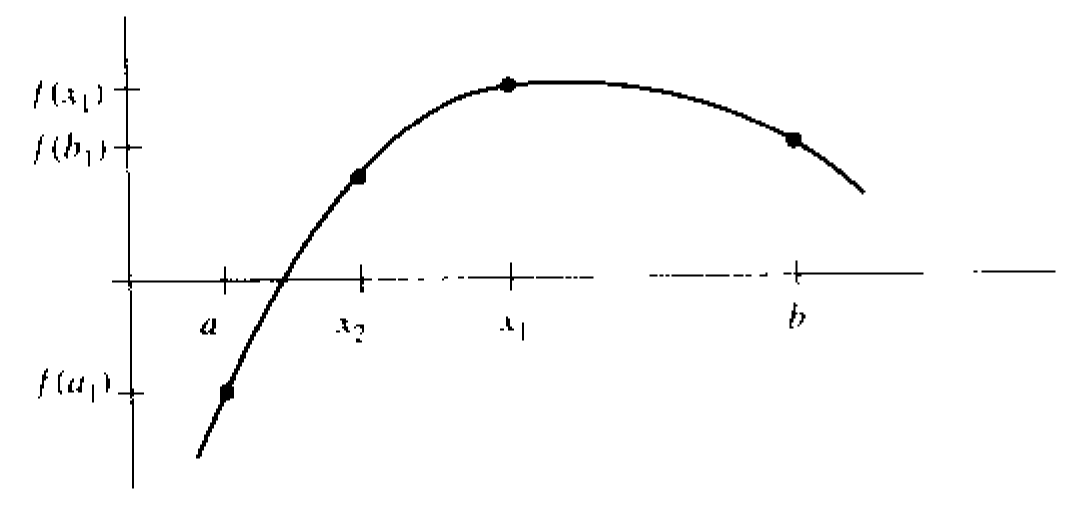


图6-9 二分法的区间序列及其前两个步骤的根

例6.6 方程 $x-x^{1/3}-2=0$ 的二分法求解

本例采用二分法求方程（6-3）的根。程序清单6-3中的函数demoBisect采用二分算法的思想实现了对函数$f(x)$的求解。后面将介绍一种更普通的二分算法。现在的重点是如何实现区间二分以及迭代过程中如何管理区间端点的替换。

254

程序清单6-3 函数demoBisect用固定次数二分迭代法求 $x-x^{1/3}-2=0$ 的根

```
function xm = demoBisect(xleft,xright,n)
% demoBisect  Use bisection to find the root of x - x^(1/3) - 2
%
% Synopsis: x = demoBisect(xleft,xright)
%           x = demoBisect(xleft,xright,n)
%
%  Input:   xleft,xright = left and right brackets of the root
%           n = (optional) number of iterations;  default:  n = 15
%
%  Output:  x = estimate of the root

if nargin<3, n=15;  end      %  Default number of iterations
a = xleft;    b = xright;    %  Copy original bracket to local variables
fa = a - a^(1/3) - 2;        %  Initial values of f(a) and f(b)
fb = b - b^(1/3) - 2;
fprintf('  k       a            xmid           b              f(xmid)\n');

for k=1:n
  xm = a + 0.5*(b-a);        %  Minimize roundoff in computing the midpoint
  fm = xm - xm^(1/3) - 2;  %  f(x) at midpoint
  fprintf('%3d  %12.8f  %12.8f  %12.8f  %12.3e\n',k,a,xm,b,fm);
  if sign(fm)==sign(fa)     %  Root lies in interval [xm,b], replace a
    a = xm;
    fa = fm;
  else                       %  Root lies in interval [a,xm], replace b
    b = xm;
    fb = fm;
  end
end
```

函数demoBisect的输入参数为区间的左端点、右端点和拟执行迭代的次数（可选参数）。

输入区间左、右端点分别存储在局部变量a和b中。$f(a)$和$f(b)$的值分别存储在fa和fb中，这样能够使$f(x)$的计算次数最少。运行函数demoBisect后的输出结果如下：

```
>> demoBisect(3,4)
  k        a             xmid            b            f(xmid)
  1    3.00000000    3.50000000    4.00000000    -1.829e-02
  2    3.50000000    3.75000000    4.00000000     1.964e-01
  3    3.50000000    3.62500000    3.75000000     8.884e-02
  4    3.50000000    3.56250000    3.62500000     3.522e-02
  5    3.50000000    3.53125000    3.56250000     8.450e-03
  6    3.50000000    3.51562500    3.53125000    -4.925e-03
  7    3.51562500    3.52343750    3.53125000     1.762e-03
  8    3.51562500    3.51953125    3.52343750    -1.582e-03
  9    3.51953125    3.52148438    3.52343750     8.959e-05
 10    3.51953125    3.52050781    3.52148438    -7.463e-04
 11    3.52050781    3.52099609    3.52148438    -3.284e-04
 12    3.52099609    3.52124023    3.52148438    -1.194e-04
 13    3.52124023    3.52136230    3.52148438    -1.490e-05
 14    3.52136230    3.52142334    3.52148438     3.735e-05
 15    3.52136230    3.52139282    3.52142334     1.123e-05

ans =
    3.5214
```

从上面的迭代过程可以看出二分法接近根的速度很慢。尽管每次迭代时，区间长度减小一半，但是f(xmid)的值不是单调变化的。例如$k = 10$处的xmid值就不如在$k = 9$处的xmid值更接近所求根，在第13次和第14次迭代也出现了相同的情况。二分法只能保证有根区间不断平分且其中包含解，但是它不能最优化地猜测出根所在区间的下一个位置。

6.3.1 二分法的分析

二分法的性能分析非常容易。假设δ_n为第n次二分后的区间长度，那么$\delta_0 = b - a$为初始区间的长度，且

$$\begin{aligned}
\delta_1 &= \frac{1}{2}\delta_0 \\
\delta_2 &= \frac{1}{2}\delta_1 = \frac{1}{4}\delta_0 \\
&\vdots \\
\delta_n &= \left(\frac{1}{2}\right)^n \delta_0
\end{aligned} \tag{6-5}$$

区间的大小可用于估算根的位置误差。由于有根区间的大小每次减少一半，所以收敛速度是线性的（即误差的减少也是线性的）。n次二分后，有根区间减小的倍数为

$$\frac{\delta_n}{\delta_0} = \left(\frac{1}{2}\right)^n = 2^{-n} \tag{6-6}$$

通过求解上面的式子能够得到迭代的次数n，即

$$n = -\log_2\left(\frac{\delta_n}{\delta_0}\right)$$

表6-1列出了一些典型n值对应的δ_n/δ_0值。并列出函数$f(x)$需要计算的次数。给定起始区间，以后每迭代一次就要计算一次函数值。

表6-1 随迭代次数增多，有根区间变小计算精度增加

n	$\dfrac{\delta_n}{\delta_0}$	函数计算次数
5	3.1×10^{-2}	7
10	9.8×10^{-4}	12
20	9.5×10^{-7}	22
30	9.3×10^{-10}	32
40	9.1×10^{-13}	42
50	8.9×10^{-16}	52

6.3.2 收敛准则

自动求根程序要反复改进根的初始估算值，才能得到比较精确的函数根。准确地说，我们无法得到精确根，因为在浮点算术中，数值程序无法得到使$f(x)=0$精确成立的x值。算法只能够判断在根的搜索结束前估算根对真实根的近似程度。

图6-10画出一个穿过x轴的函数。测试收敛性可以采用两个准则：第一个是测试相邻两次迭代估算根之间的差值，这要求用户为x值的可接受程度确定一个容差；第二个是测试当前函数$f(x)$的大小，这要求用户给定$f(x)$的容差。x和$f(x)$的容差不一定相同，此外，不能保证在任何情况下这两个准则都是一个好的退出计算的判断条件。当$f(x)$比较平稳时（如，$|df/dx|$接近根），x的容差就会很保守；当函数$f(x)$变化剧烈时，$f(x)$的容差就会很保守。

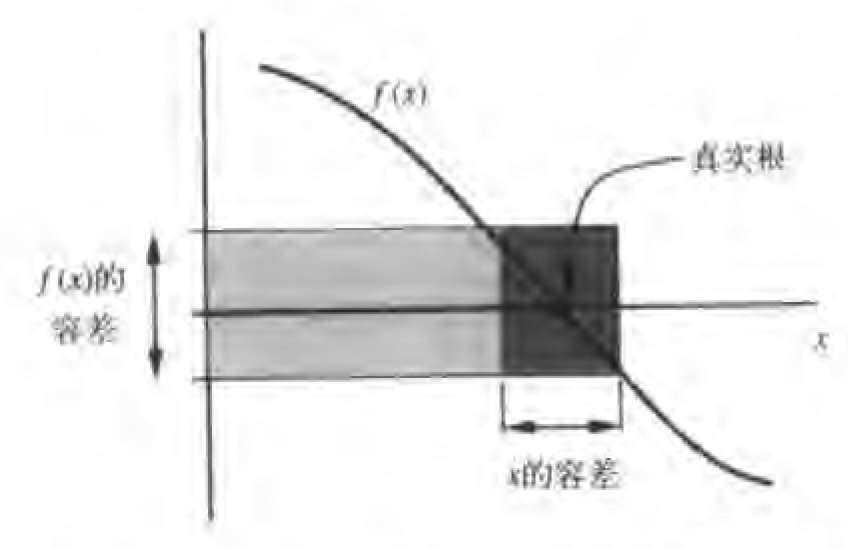


图6-10 求根程序的终止准则应该既考虑到x的容差，也考虑到$f(x)$的容差

257

用容差来终止寻根程序运行有两个重要原因。第一，在实际中，可能不需要方程根有很多的有效位。第二，由于计算机只能表示有限的精度，因此所求根已经不能继续细化。另外，由于舍入误差的影响，用x的真实根代入$f(x)$得到的结果也不一定为零。

假设根最近的两个估算值为x_k和x_{k-1}，它们是最新的有根区间的两个端点值，当出现

$$|x_k - x_{k-1}| < \varepsilon_m \tag{6-7}$$

的情况时，继续迭代就变得没有意义。这里ε_m是机器精度，也是求根精度的绝对极限。当$x\approx1$时，虽然此容差也有限制，可是它仍然会有效。但是，当$|x_k|>>1$时，由于差值x_k-x_{k-1}的舍

入误差总是大于ε_m，所以方程（6-7）中的条件永远不会成立。下面是一个较好的收敛判断标准

$$\left|x_k - x_{k-1}\right| < \delta_x \left|b - a\right| \tag{6-8}$$

其中$[a,b]$是起始区间，δ_x是一个很小但比ε_m大的数。方程（6-8）等价于

$$\left|\frac{x_k - x_{k-1}}{b-a}\right| < \delta_x \tag{6-9}$$

它表明了此测试测量了原始区间相对缩小的程度。

当前根的估算值所对应的函数值足够小时，求根迭代也应该终止。这可以由下面的收敛准则实现：

$$\left|f(x_k)\right| < \delta_f \max\left\{\left|f(a_0)\right|,\left|f(b_0)\right|\right\} \tag{6-10}$$

258

其中a_0和b_0是根所在的起始区间的端点，δ_f是函数$f(x)$在根上的相对容差。通常，方程（6-8）和（6-10）在判定程序终止时都要用到。一种指定δ_x和δ_f值的方法是使用ε_m的倍数。严格的收敛准则中$\delta_x \approx 5\varepsilon_m$，$\delta_f \approx 5\varepsilon_m$。实际上，$\delta_x$和$\delta_f$可以取不同的值。

6.3.3 二分法的一般实现

程序清单6-4中的函数bisect实现的二分算法包含了上节提到的两个终止准则，此函数有下列4种调用形式：

```
r = bisect(fun,xb)
r = bisect(fun,xb,xtol)
r = bisect(fun,xb,xtol,ftol)
r = bisect(fun,xb,xtol,ftol,verbose)
```

其中*fun*是计算$f(x)$的m文件的名字，*xb*是行向量，它包含起始有根区间端点，*xtol*是方程（6-9）中的δ_x的值，*ftol*是方程（6-10）中δ_f的值，*verbose*是一个标志，用来控制是否打印二分迭代的中间结果。*xb*的第一个元素是区间的左端点，第二个元素是区间的右端点。根的预测返回值赋给*r*。

给定程序清单6-2中的函数fx3后，结合函数brackPlot和函数bisect可以得到方程（6-3）的根：

```
>> xb = brackPlot('fx3',0,5);
>> bisect('fx3',xb,5e-5)
ans =
    3.5214
```

把$xtol = 5\times10^{-5}$提供给函数bisect，使迭代次数少于50次。*xtol*的这个值意味着相对于初始有根区间，返回根的前4个有效位是正确的（参见表6-1）。

要使用标志位verbose，就要求指定*xtol*和*ftol*的值。例如，输入下列语句重新产生例6.6的结果：

```
>> bisect('fx3',[3 4],5e-5,5e-6,1)

Bisection iterations for fx3.m
   it        xm           fm
   1     3.5000e+00   -1.8294e-02
   2     3.7500e+00    1.9638e-01
   3     3.6250e+00    8.8842e-02
```

```
  14     3.5214e+00     3.7349e-05
  15     3.5214e+00     1.1227e-05
  16     3.5214e+00    -1.8347e-06
ans =
   3.5214
```

我们刚刚介绍了二分法如何结合brackPlot函数和bisect函数来求方程的单个根。同样，对于有多个根的方程，只要将同样的程序加以扩展即可用来求解方程的多个根。考虑下列代码：

```
xmin = ... ; xmax = ... ,
Xb = brackPlot('myFunction',xmin,xmax);
for k = 1:size(Xb,1)
    x(k) = bisect('myFunction',Xb(k,:));
    fprintf('Suspected root at %f gives f(x) = %f/n',...
             x(k),myFunction(x(k)));
end
```

表达式size(Xb,1)返回Xb的行数。既然Xb的每一行都包含了一个有根区间，size(Xb,1)的意思就是在区间xmin≤x≤xmax上找到的有根区间数。For循环用语句Xb(k,:)将Xb的第k行的值传给bisect函数。本章的后面会介绍其他求根程序，它们可以代替bisect函数用来求根

程序清单6-4　采用二分法求根的MATLAB函数bisect

```
function r = bisect(fun,xb,xtol,ftol,verbose)
% bisect  Use bisection to find a root of the scalar equation f(x) = 0
%
% Synopsis:  r = bisect(fun,xb)
%            r = bisect(fun,xb,xtol)
%            r = bisect(fun,xb,xtol,ftol)
%            r = bisect(fun,xb,xtol,ftol,verbose)
%
% Input: fun     = (string) name of function for which roots are sought
%        xb      = vector of bracket endpoints. xleft = xb(1), xright = xb(2)
%        xtol    = (optional) relative x tolerance.    Default:  xtol=5*eps
%        ftol    = (optional) relative f(x) tolerance. Default:  ftol=5*eps
%        verbose = (optional) print switch. Default: verbose=0, no printing
%
% Output:  r = root (or singularity) of the function in xb(1) <= x <= xb(2)
if size(xb,1)>1, warning('Only first row of xb is used as bracket');  end
if nargin < 3,  xtol = 5*eps;  end
if nargin < 4,  ftol = 5*eps;  end
if nargin < 5,  verbose = 0;   end

 xeps = max(xtol,5*eps);          %  Smallest tolerances are 5*eps
 feps = max(ftol,5*eps);
 a = xb(1,1); b = xb(1,2);        %  Use first row if xb is a matrix
 xref = abs(b - a);               %  Use initial bracket in convergence test
 fa = feval(fun,a);   fb = feval(fun,b);
 fref = max([abs(fa) abs(fb)]);  %  Use max f in convergence test
 if sign(fa)==sign(fb)            %  Verify sign change in the interval
   error(sprintf('Root not bracketed by [%f, %f]',a,b));
 end

 if verbose
   fprintf('\nBisection iterations for %s.m\n',fun);
   fprintf('   k       xm             fm\n');
```

```
end
k = 0;  maxit = 50;        % Current and max number of iterations
while k < maxit
  k = k + 1;
  dx = b - a;
  xm = a + 0.5*dx;         % Minimize roundoff in computing the midpoint
  fm = feval(fun,xm);
  if verbose, fprintf('%4d  %12.4e  %12.4e\n',k,xm,fm); end

  if (abs(fm)/fref < feps) | (abs(dx)/xref < xeps) % True when root is found
    r = xm;  return;
  end
  if sign(fm)==sign(fa)
    a = xm;  fa = fm;      % Root lies in interval [xm,b], replace a and fa
  else
    b = xm;  fb = fm;      % Root lies in interval [a,xm], replace b and fb
  end
end
warning(sprintf('root not within tolerance after %d iterations\n',k));
```

260

最后，提醒读者要检验自动求根程序所得到的结果。如果bisect返回的x(k)是所求根，那么输出语句fprintf中myFunction(x(k))的值应足够接近零才行。

6.4 牛顿法

牛顿法也叫做牛顿—拉夫申（Newton-Raphson）法，它利用函数$f(x)$及其一阶导数来求根。图6-11描述了牛顿法的两步迭代。先得到第一个近似根为x_1，但要用其他方法得到这个根，比如二分法、区间划分法或猜测法等。然后在x_1处计算$f(x_1)$和一阶导数$f(x_1)$的值，在几何上，$f(x_1)$是$f(x)$在$x=x_1$处的切线，这条切线经过x轴的点为第二个近似根x_2。

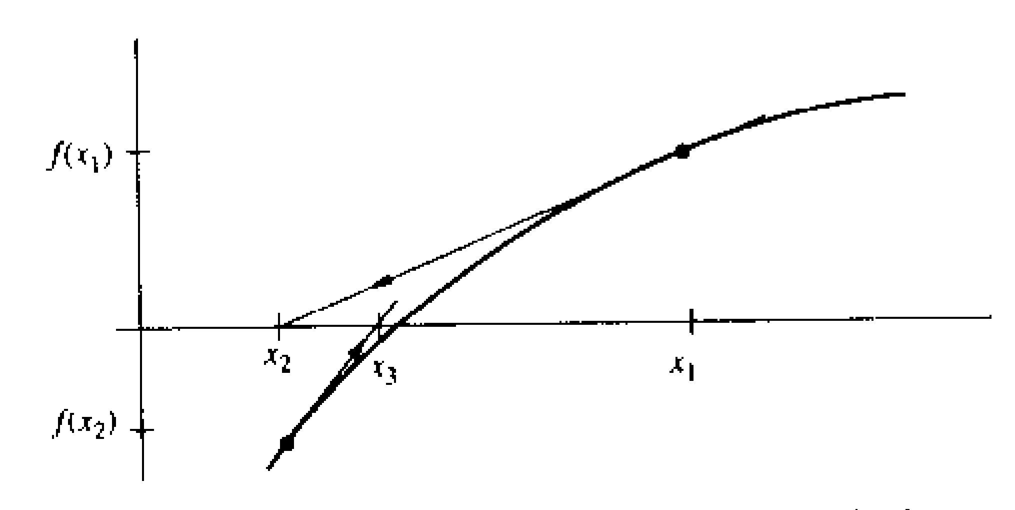


图6-11　牛顿法经过两步迭代后得到的近似根序列

根据泰勒级数对$f(x)$的近似得到计算公式。给定x_k，其中k是近似根序列的下标

261

$$f(x_k+\Delta x)=f(x_k)+\Delta x\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_k}+\frac{(\Delta x)^2}{2}\left.\frac{\mathrm{d}^2f}{\mathrm{d}x^2}\right|_{x_k}+\cdots \tag{6-11}$$

上式中，Δx是x_k与所期望近似根之间的距离。把$\Delta x=x_{k+1}-x_k$代入方程（6-11），只保留级数展开式的第一项，得：

$$f(x_{k+1})=f(x_k)+(x_{k+1}-x_k)f'(x_k)$$

其中

$$f'(x_k)=\left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_{x_k}$$

计算的目的是找出满足$f(x)=0$的x值。令$f(x_{k+1})=0$，解出x_{k+1}得：

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)} \tag{6-12}$$

给定x_k值和函数的解析表达式，很容易计算方程（6-12）的右部。用牛顿法求根的算法很简洁，因为牛顿法不需要维持有根区间。算法如下所示：

算法6.4 标量方程的牛顿法求解

```
initialize: x_1=...
for k = 2,3,...
    x_k = x_{k-1} - f(x_{k-1})/f'(x_{k-1})
    if converged, stop
end
```

和二分法不同，牛顿法不在x轴上的一段有限范围内求根，所以，求根的迭代次数与初始猜测关系不大，在最坏的情况下这可能会导致迭代发散或浮点计算异常[⊖]。如果$f(x)$的一阶导数成为零，就无法计算x_{k+1}，如图6-12所示，这样尽管初始猜测值可能很准确，但如果当前估算值附近函数的一阶导数趋近零，牛顿法也可能发散。然而，一旦牛顿法收敛，它的迭代速度就会比二分算法快很多。



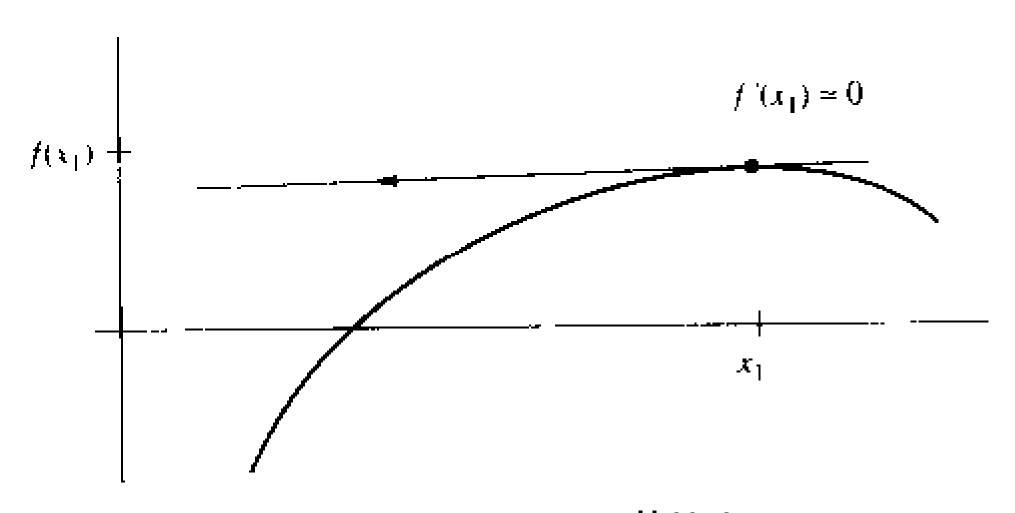


图6-12 牛顿法的缺陷

牛顿法是定点迭代的一种形式，其迭代函数为

$$g(x)=x-\frac{f(x)}{f'(x)}$$

例6.4和例6.5中的$g_3(x)$就是根据上式得来的。

例6.7 应用牛顿法计算$x-x^{1/3}-2=0$的根

程序清单6-5中的函数demoNewton利用牛顿法求方程（6-3）的根，代码中直接包含了$f(x)$和$f'(x)$的求解部分。下节将介绍牛顿法的一种更灵活的实现方式。

demoNewton对初始猜测值$x=3$的运行结果如下所示：

```
>> demoNewton(3)
  k      f(x)          dfdx          x(k+1)
  0   -4.422e-01    8.398e-01    3.526644293139033
  1    4.507e-03    8.561e-01    3.521380147397328
```

⊖ 计算结果为NaN或Inf。

```
  2     3.771e-07     8.560e-01     3.521379706804571
  3     2.665e-15     8.560e-01     3.521379706804568
  4     0.000e+00     8.560e-01     3.521379706804568

ans =
   3.5214
```

263

程序清单6-5　函数demoNewton用固定次数牛顿迭代法求方程$x - x^{1/3} - 2=0$的根

```
function x = demoNewton(x0,n)
% demoNewton  Use Newton's method to find the root of f(x) = x - x^(1/3) - 2
%
% Synopsis:  x = demoNewton(x0)
%            x = demoNewton(x0,n)
%
%  Input:    x0 = initial guess
%            n = (optional) number of iterations, Default: n = 5
%
%  Output    x = estimate of the root

if nargin<2, n=5; end        %  Default number of iterations
x = x0;                      %  Initial guess

fprintf('  k        f(x)           dfdx            x(k+1)\n');
for k=1:n
  f = x - x.^(1/3) - 2;
  dfdx = 1 - (1/3)*x.^(-2/3);
  x = x - f/dfdx;
  fprintf('%3d  %12.3e  %12.3e   %18.15f\n',k-1,f,dfdx,x);
end
```

牛顿法的收敛速度特别快，它经过3次迭代后就可以精确到根的前16位，并在机器精度上满足$f(x) = 0$。相比之下，二分法迭代经过15次迭代后只能精确到根的前5位。

6.4.1　牛顿法的收敛性

若$f(x)$、$f'(x)$、$f''(x)$连续，且$f'(x)$和$f''(x)$在根的一个邻域内非零，就可以得到一个解析表达式来描述牛顿法的快速收敛性（参见文献[10，66，68]）。

假设ξ是$f(x)$的精确根。如果初始猜测值x_0足够接近根（也就是说，存在某个正数ρ，有$|x_0 - \xi| < \rho$），那么，对于以后每次使用方程（6-12）的迭代，都有$|x_k - \xi| < \rho$，且

$$|x_{k+1} - \xi| \leqslant C(\rho)|x_k - \xi|^2$$

其中$C(\rho)$是独立于k的因子。上式说明每次迭代的误差$e_{k+1} = x_{k+1} - \xi$是上一次迭代误差的平方，由此，称牛顿法有二次收敛性(*quadratic convergence*)，相比之下，二分法只是一次(线性)收敛（参见方程（6-5））。

264

牛顿法的二次收敛是有限制的。首先，只有满足$|x_0 - \xi| < \rho$时，牛顿法才会收敛，通常情况下，ξ和ρ都是未知的。换句话说，当初始猜测值足够接近真值时，牛顿法才能收敛，但是无论是根还是“足够接近”的标准都无法提前预知。第二，出现重根时，有$f'(\xi)=0$，此时前面提到的收敛公式不再适用。出现重根时，牛顿法是线性收敛而不是二次收敛的。

对良性（well-behaved）函数来说，可以利用牛顿法的快速收敛性。而对于一般的求根程

序，就需要健壮性更好的解法。一种改进收敛性的方法是使用混合法，混合法结合了二分法和牛顿法的优点（参照6.6节）。

6.4.2 牛顿法的一般实现

前面的例子示范了牛顿法在对指定的函数$f(x)$求根中的应用。但是，如果用它来求其他函数的根，就需要重编写代码。程序清单6-6中的函数newton可以用牛顿法对任意函数$f(x)$求根，它有下列4种调用形式：

```
r = newton(fun,x0)
r = newton(fun,x0,xtol)
r = newton(fun,x0,xtol,ftol)
r = newton(fun,x0,xtol,ftol,verbose)
```

第一个参数fun用来指定在给定x值的情况下求$f(x)$和$f'(x)$值的函数名；第二个参数x0是初始猜测值；xtol和ftol分别是x的绝对容差和$f(x)$的绝对容差；verbose是标志位，指明是否打印迭代的中间结果；输出r是估算根。

由于没有起始有根区间，因此很难计算相对收敛容差，检测收敛性相当复杂。程序清单6-6中的程序要求用户给出x和$f(x)$的绝对容差，将它与程序清单6-4中的bisect函数中使用的相对容差作个比较。

函数newton的第一个参数是由用户给出的函数名，用来计算$f(x)$和$f'(x)$的值，例6.8提供了如何创建这种函数的细节。

用户可以手工编写程序，也可以使用更一般性的newton函数来求根，这是喜好和习惯问题。由于newton函数经过调试，而且能正确地对任何用户自定义函数用牛顿法求根，所以建议用户使用newton函数。另外，函数newton还使用了一种灵活和更一般的方式求收敛容差。另一方面，由于牛顿迭代公式很简洁，我们可以在算法中包含$f(x)$和$f'(x)$的计算。用户编写的牛顿迭代法求根的好处就是可以将所有的代码写在一个文件里。

265

程序清单6-6 用牛顿法求根的newton函数

```
function r = newton(fun,x0,xtol,ftol,verbose)
% newton     Newton's method to find a root of the scalar equation f(x) = 0
%
% Synopsis:  r = newton(fun,x0)
%            r = newton(fun,x0,xtol)
%            r = newton(fun,x0,xtol,ftol)
%            r = newton(fun,x0,xtol,ftol,verbose)
%
% Input:  fun     = (string) name of mfile that returns f(x) and f'(x).
%         x0      = initial guess
%         xtol    = (optional) absolute tolerance on x.    Default: xtol=5*eps
%         ftol    = (optional) absolute tolerance on f(x). Default: ftol=5*eps
%         verbose = (optional) flag.  Default: verbose=0, no printing.
%
% Output.  r = the root of the function

if nargin < 3,  xtol = 5*eps;   end
if nargin < 4,  ftol = 5*eps;   end
if nargin < 5,  verbose = 0;    end
xeps = max(xtol,5*eps);  feps = max(ftol,5*eps);  % Smallest tols are 5*eps
```

```
if verbose
  fprintf('\nNewton iterations for %s.m\n',fun);
  fprintf('  k      f(x)          dfdx          x(k+1)\n');
end

x = x0;  k = 0;  maxit = 15;     %  Initial guess, current and max iterations
while k <= maxit
  k = k + 1;
  [f,dfdx] = feval(fun,x);       %   Returns f( x(k-1) ) and f'( x(k-1) )
  dx = f/dfdx;
  x = x - dx;
  if verbose, fprintf('%3d %12.3e %12.3e %18.14f\n',k,f,dfdx,x);  end

  if ( abs(f) < feps ) | ( abs(dx) < xeps ),  r = x;  return;  end
end
warning(sprintf('root not found within tolerance after %d iterations\n',k));
```

266

例6.8　newton函数的应用

用newton函数计算方程（6-3）的根时，还需要一个m文件函数计算$f(x)$和$f'(x)$的值。程序清单6-7中的函数fx3n可以计算$f(x)$和$f'(x)$并把结果分别返回给变量f和dfdx。

程序清单6-7　用fx3n函数求$f(x)$和$f'(x)$的值，其中$f(x) = x - x^{1/3} - 2$，返回值为一个与newton函数兼容的行向量

```
function [f,dfdx] = fx3n(x)
% fx3n  Evaluate f(x) = x - x^(1/3) - 2 and dfdx for Newton algorithm
f    = x - x.^(1/3) - 2;
dfdx = 1 - (1/3)*x.^(-2/3);
```

令初始猜测值$x = 3$，将fx3n函数应用于newton函数，运行结果如下：

```
>> newton('fx3n',3,5e-16,5e-16,1)

Newton iterations for fx3n.m
  k      f(x)          dfdx          x(k+1)
  1   -4.422e-01    8.398e-01    3.52664429313903
  2    4.507e-03    8.561e-01    3.52138014739733
  3    3.771e-07    8.560e-01    3.52137970680457
  4    2.665e-15    8.560e-01    3.52137970680457
  5    0.000e+00    8.560e-01    3.52137970680457

ans =
    3.5214
```

verbose标志为真，所以将中间结果打印出来。参数xtol和ftol都设为很小的数，以便和demoNewton例子的结果作比较。

正如我们所期望的，最后结果与例6.7中的函数demoNewton计算结果一致。

例6.9　用牛顿法设计野餐桌的桌腿

用牛顿法解决设计野餐桌的桌腿问题（参见例6.1）需要采用与前例相同的几个步骤：编写m文件计算$f(\theta)$和$f'(\theta)$，把m文件名和初始猜测值传给newton函数。根据方程（6-2）得到辅助m文件中所使用的公式为：

$$f(\theta)=w\sin\theta-h\cos\theta-b \qquad f'(\theta)=w\cos\theta+h\sin\theta$$

267

用m文件实现 $f(\theta)$ 和 $f'(\theta)$ 的计算时，需要考虑如何给参数 w、h 和 b 赋值。函数newton的已知量只是独立变量值（本例中指 θ）、函数值及其一阶导数的值，它没有额外参数与计算 $f(\theta)$ 和 $f'(\theta)$ 的m文件函数通信（本例最后介绍newton函数的另一种实现形式）。

一个简单的方法就是把指定的 w、h 和 b 的值硬编码到辅助m文件中。显然，每次 w、h 和 b 的值变化后，都需要重写辅助m文件。另一种方法就是不用newton函数，而为特定问题编写一个定制的用牛顿法实现的程序，它在同一个m文件中计算 $f(\theta)$ 和 $f'(\theta)$ 的值并实现牛顿法迭代（参见例6.7和习题20）。第三种方法，也是我们在这里采用的方法，就是使用全局变量通过Newton函数将 w、h 和 b 传给辅助m文件中加以实现。关于MATLAB中全局变量的介绍请参照3.6.2节。

程序清单6-8中的tablen（主）程序和legsn（辅助）程序用来计算桌子的尺寸。桌腿的参数 w、h 和 b 的值通过全局变量WLENGTH、HLENGTH和BLENGTH传给legsn程序。为避免变量重名，全局变量使用特殊的名称（而不是 w、h 和 b）并全用大写字母来表示。另外，根据传统，MATLAB函数输入变量的值不能改变，这些全局变量要从newton函数的输入参数中拷贝过来。全局变量和共享它们的函数之间的关系如图6-13所示。

运行tablen得到的结果如下所示：

```
>> tablen(28,27,3.5);
theta =   49.1 degrees   d1 =   10.782   d2 =   17.855
```

若重新编写newton函数，附设额外参数传给用户自定义函数 $f(x)$，也可以不使用全局变量。有兴趣的读者可以参考NMM工具箱中的newtonNG、tablenNG和legsNG等m文件。函数newtonNG使用varargin工具（程序）把参数 w、h 和 b 的值传给legsNG函数。参照例6.11中看这一技术是如何与内置函数fzero共同起作用的。

268

程序清单6-8　设计野餐桌桌腿的函数tablen和函数legsn。求根要利用程序清单6-6中的newton函数

```
function tablen(w,h,b,t0)
% tablen  Use Newton's method to find dimensions of picnic table legs
%
% Synopsis:  tablen(w,b,h)
%            tablen(w,b,h,t0)
%
% Input:     w,h = width and height of the table legs
%            b   = width of the material used to make the legs
%            t0  = (optional) initial guess at theta.  Default: t0 = pi/4 (rad)
%
% Output:    Print out table dimensions

global WLENGTH HLENGTH BLENGTH              %  Define global variables
WLENGTH = w;  HLENGTH = h;  BLENGTH = b;   %  Global copies of input variables

if nargin<3
   error('All three dimensions, w, h, b, must be specified');
elseif nargin<4
   t0 = pi/4;          % default initial guess
end

theta = newton('legsn',t0);       %  Root-finding is done in newton
```

```
% --- Compute other dimensions once theta is known
alpha = pi/2 - theta;      a  = BLENGTH/tan(alpha);      c = BLENGTH/tan(theta);
d2 = HLENGTH/(2*sin(theta));     d1 = d2 - a - c;
fprintf('theta = %6.1f degrees   d1 = %8.3f   d2 = %8.3f\n',theta*180/pi,d1,d2);

function [f,dfdt] = legsn(theta)
% legsn  Evaluate f(theta) and fprime(theta) for the picnic leg problem.
%        Used with the Newton's method.
%
% Synopsis:  [f,dfdt] = legsn(theta)
%
% Input:     theta = angle of table leg in radians
%            WLENGTH HLENGTH and BLENGTH are global variables giving
%                    table dimensions
%
% Output:    f    = value of f(theta). As theta approaches a root,
%                   f(theta) approaches zero.
%            dfdt = fprime(theta), ie. df/dtheta
global  WLENGTH  HLENGTH  BLENGTH

f    = WLENGTH*sin(theta) - HLENGTH*cos(theta) - BLENGTH;
dfdt = WLENGTH*cos(theta) + HLENGTH*sin(theta);
```

269

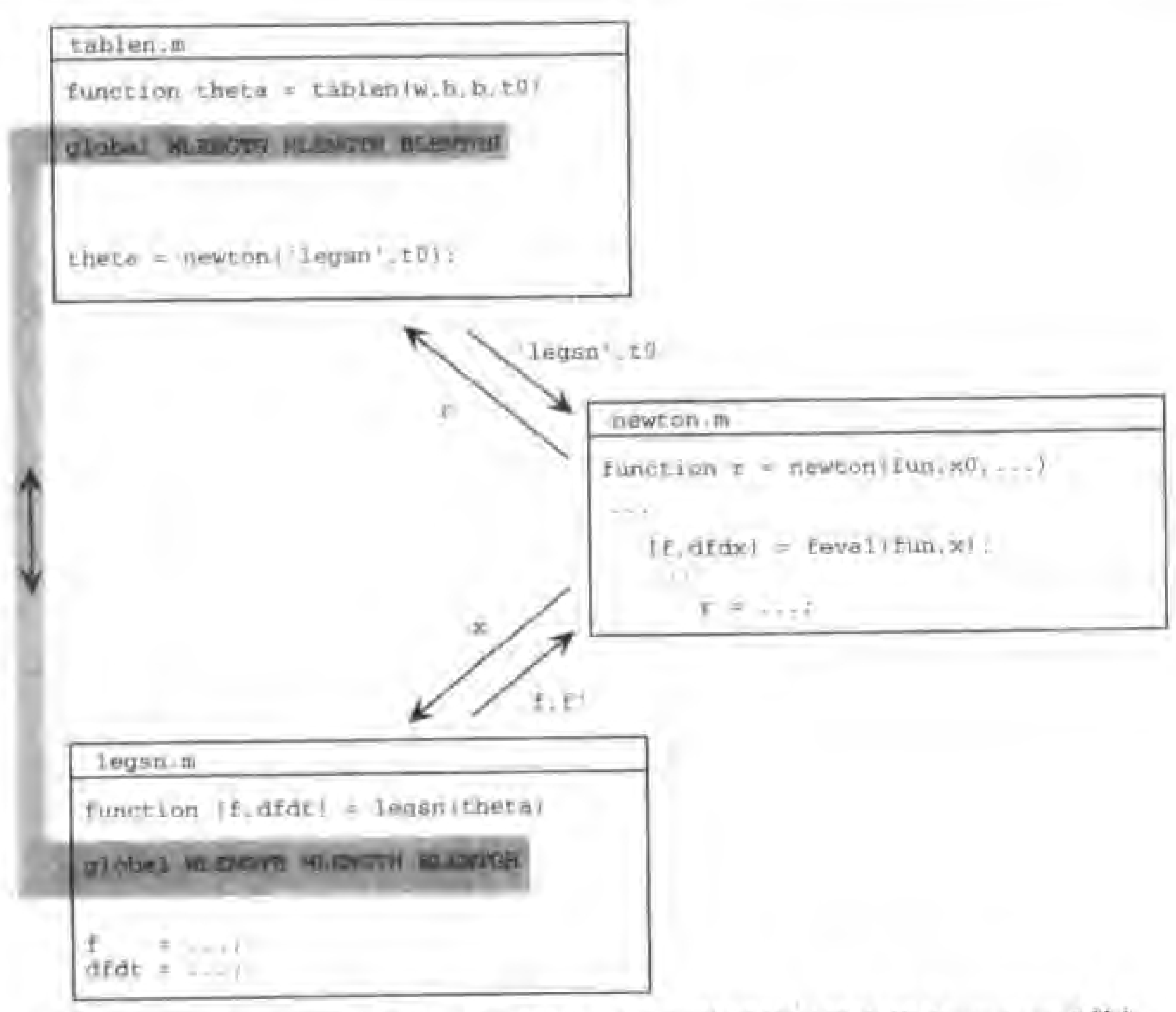


图6-13 使用全局变量实现主函数与辅助函数间的通信（没有使用newton函数）

6.5 割线法

割线法（secant method）和牛顿法类似，也要用到函数在当前近似根附近的斜率。但是

它不使用理论公式求$f'(x)$，而是利用最后两个根的近似值来进行斜率计算。已知x_k、$f(x_k)$、x_{k-1}和$f(x_{k-1})$，则$f(x)$的一阶导数近似为

$$f'(x_k) \approx \frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}$$

把上式代入方程（6-12）得到

$$x_{k+1} = x_k - f(x_k)\left[\frac{(x_k - x_{k-1})}{f(x_k)-f(x_{k-1})}\right] \tag{6-13}$$

割线法需要两个初始近似根，这和二分法一样。这两个近似根可以是有根区间的两个端点或其他求根程序两次迭代的结果。与二分法不同的是，割线法不会把查找范围限制在起始有根区间中。割线法不断地修正近似根，不论这些值在有根区间内还是有根区间外，都是有意义的。

270

割线算法描述如下：

算法6.5 割线法

```
initialize: x1=..., x2=...
for k = 2,3...
    x(k+1) = x(k) - f(x(k))(x(k) - x(k-1)) / (f(x(k)) - f(x(k-1)))
    if converged, stop
end
```

图6-14简略地列出了几步迭代序列。假设x_1和x_2是由区间划分法获得的两个值，第三个估算根x_3在区间中，而第四个却在区间外。

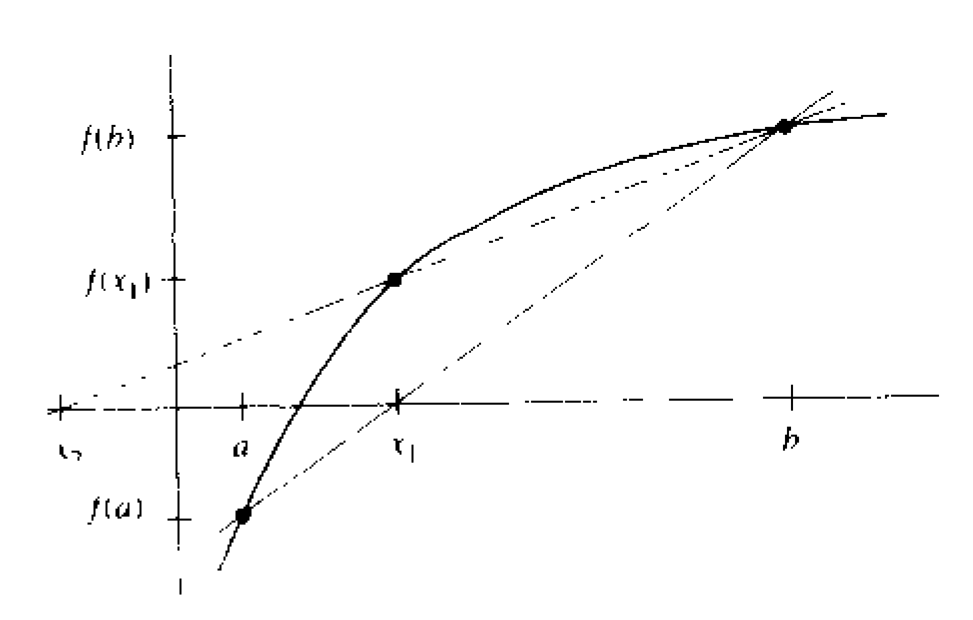


图6-14 割线法得到的近似根序列

一般很容易将方程（6-13）变换为

$$x_{k+1} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k)-f(x_{k-1})} \tag{6-14}$$

这里并不建议读者采用这种变换方式。尽管方程（6-13）和（6-14）在算术上等价，但是方程（6-13）更适合进行数值计算。当迭代结果接近到收敛值时，方程（6-13）和（6-14）的区别就很明显了，理想状态下，有：

$$\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \to \frac{1}{f'(\xi)} \quad 当\ x_k \to \xi$$

其中ξ是所求根。但是由于舍入误差的影响，$f(x_k)-f(x_{k-1})$的数值结果可能为零，即使不为零，当$f(x_k)$和$f(x_{k-1})$的值变得更接近时，也会导致精度丢失。这种精度丢失会导致严重的消去错误，因为分母中很小的误差都会导致$(x_k - x_{k-1})/(f(x_k)-f(x_{k-1}))$的计算结果出现相当大的误差。

由于计算$f(x_k)-f(x_{k-1})$时的舍入误差，方程（6-13）和（6-14）都可能出现严重的消去错误。方程（6-14）中，消去错误会直接影响x_{k+1}的结果。方程（6-13）可以写为

271

$$x_{k+1} = x_k + \Delta x$$

其中

$$\Delta x = f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$$

当迭代快要收敛时，舍入误差会损坏$(x_k - x_{k-1})/(f(x_k)-f(x_{k-1}))$的值，使它接近一个常量，这个值可能接近，也可能不接近$1/f'(\xi)$。只要$(x_k - x_{k-1})/(f(x_k)-f(x_{k-1}))$的值不是无穷大（参照习题27），它的具体值就不重要，因为它与$f(x_k)$相乘，而$f(x_k)$在收敛时接近于零。可见Δx是个很小的数，它的值不受$(x_k - x_{k-1})/(f(x_k)-f(x_{k-1}))$的舍入误差的影响。

如图6-15所示，当x_k和x_{k-1}差距不大，而且$f(x_k)\approx f(x_{k-1})$时，方程（6-13）同样容易出现问题。当$f'(x)\approx 0$时，割线法和牛顿法的缺陷是一样的，都会发散。

割线法的MATLAB实现留作课后练习。

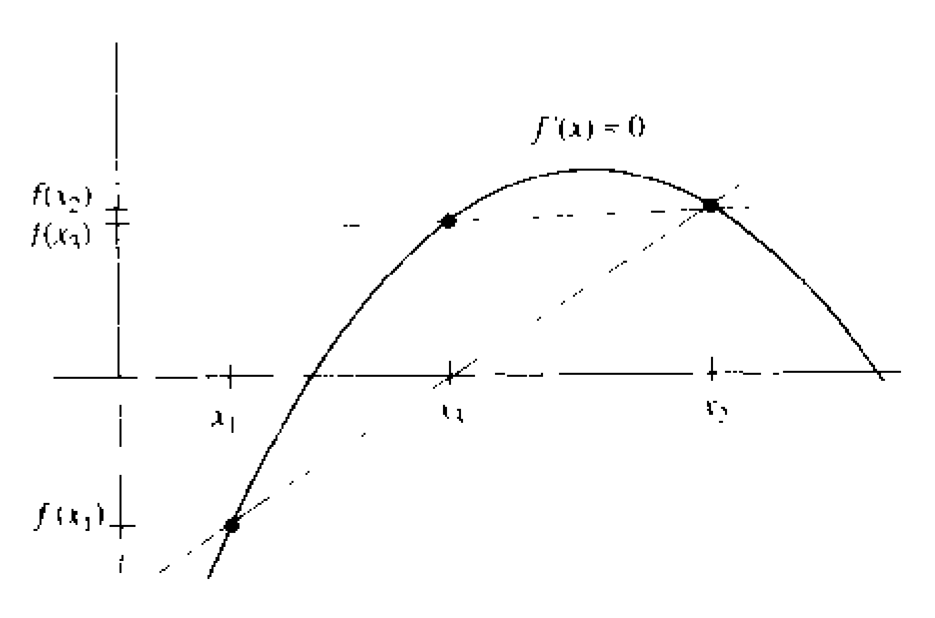


图6-15 割线法的缺陷

例6.10 用割线法求方程$f(x) = x - x^{1/3} - 2=0$的根

用割线法解方程（6-3）得到下列结果：

k	x_{k-1}	x_k	$f(x_k)$
0	4	3	4.4225×10^{-1}
1	3	3.51734262	3.4555×10^{-3}
2	3.51734262	3.52141665	3.1625×10^{-5}
3	3.52141665	3.52137970	-2.0347×10^{-4}

（续）

k	x_{k-1}	x_k	$f(x_k)$
4	3.52137959	3.52137971	-1.3323×10^{-15}
5	3.52137971	3.52137971	-2.2204×10^{-16}
6	3.52137971	3.52137971	0

272

此计算的初始猜测值是$x=3$和$x=4$。割线法的收敛速度比牛顿法稍慢，却比二分法快很多。

6.6 混合法

到现在为止，介绍的求根法都是直接使用迭代算法，它们要么很健壮（如二分法），要么收敛速度很快（如牛顿法和割线法）。如果以程序的逻辑复杂性为代价，可以将这些方法结合使用。参考文献[43、61]介绍了使用混合法编写的Fortran和C程序。

混合法结合了二分法和另一个收敛更快的方法，如牛顿法或割线法。每次迭代首先用快速迭代法的基本步骤，若结果在起始有根区间中，就继续使用这一方法，否则就采用二分法。这种混合法不比二分法的收敛速度慢，有些情况下还会快很多。它的优点是可以给出粗略的猜测值而不必担心所用方法会因此发散。本章的课后练习将讨论混合法的实现。

fzero函数

MATLAB函数fzero实现了混合法，它结合了二分法、割线法和二次反插值法（inverse quadratic interpolation）。如图6-16所示，它的每次迭代都需要三对（$x, f(x)$）数据。假设它们分别为（$a, f(a)$）、($b, f(b)$)和($c, f(c)$)，先用二次反插值法求出二次插值式穿过x轴的点x_{quad}；如果这三点不是明显分开的（即$a\approx b$或$b\approx c$），就用割线法求x_{lin}，它是线性插值式经过x轴的点；如果x_{quad}和x_{lin}都不在原始的有根区间中，就采用二分法。

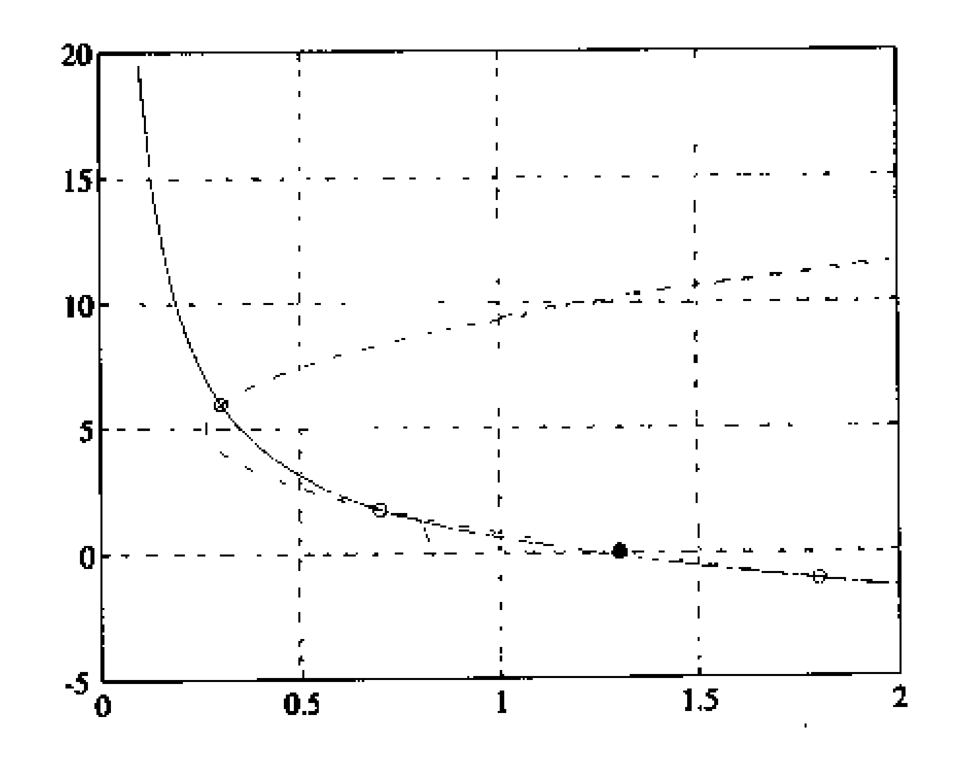


图6-16 对$f(x)=2/x-x-0.4$进行二次反插值（实线）。空心圆描述的三对（x, y）数据对用来构造二次反插值式$x=\alpha_1 y^2+\alpha_2 y+\alpha_3$（虚线）。二次插值式与坐标轴的交点（实心点）作为下一个近似根

273

fzero函数中包含的算法比上面描述的过程要复杂（精细）得多，采用了避免冗余计算并将舍入误差产生的影响减到最小的措施。有兴趣的读者除了阅读fzero.m中的代码，还可

以参阅文献[8、24、61]。另外，Forsythe等人在文献[24]中介绍了用Fortran语言实现的fzero函数；Press等人在文献[61]中介绍了用C语言的实现的fzero函数。这一算法首先由van Wijingaarden和Dekker开发，后来由Brent [8]改进推广。

从MATLAB 4.0到MATLAB 5.3，函数fzero的语法随着版本变化有所改进，这里我们只介绍5.3版本的语法，旧版本的用户请参考文档资料，并要特别注意控制fzero函数执行的选项的用法。

函数fzero(5.3版本)有很多种调用形式，下面是其中的3种基本调用形式[⊖]：

```
r = fzero(fun,x0)
r = fzero(fun,x0,options)
r = fzero(fun,x0,options,arg1,arg2,    )
```

其中*fun*是用来计算$f(x)$的m文件的名字，*x0*是初始猜测值，它或为标量或为由两个元素组成的向量。若*x0*为标量，fzero首先进行基本搜索来找到一个有根区间，找到后就执行主算法；若*x0*为向量，它的两个元素构成的区间中必须有根，即$f(x)$在$x_0(1)$和$x_0(2)$处的函数值符号相反。

$f(x)$的初始猜测值可以通过观察得到，也可由brackPlot函数自动完成。例如，下面的语句求出方程（6-3）的根（也可参照程序清单6-2）：

274

```
>> xb = brackPlot('fx3',0,5);
>> fzero('fx3',xb)
ans =
    3.5214
```

参数*options*是由函数optimset创建的一种特殊数据结构，其具体使用将在下一段介绍。附加参数*arg1,arg2,...*不在fzero函数中使用，而是被传递给*fun*指定的函数，这些传递参数（*pass-through parameter*）用于指定$f(x)$中各项的系数，可以避免使用全局变量。例6.11示范了这种方法的具体实现。

参数*options*的使用是MATLAB 5.3版本的新特性。以前的版本只有两个选项：收敛容差和运行函数fzero时的打印控制标志。函数optimset是一种工具程序，它允许为各种优化函数单独设置选项。有些优化函数在MATLAB标准工具箱中（如fzero和fmins），其他的在优化工具箱中，这一工具箱要向MathWorks公司单独购买。由于optimset是许多函数的接口，因此使用help optimset得到的文档资料相当多，不利于用户阅读。然而，在使用fzero函数时，只用到optimset指定的几个参数。

为了指定控制fzero的可选参数，用户首先需要用optimset创建一个*options*数据结构，然后把这个新建的数据结构作为fzero的第三个参数来传递。下面就是一种典型应用：

```
>> x0 = ...
>> options = optimset( ... )
>> x = fzero('myFun',x0,options)
```

函数optimset的语法为：

```
options = optimset('parameterName',parameterValue,...)
```

其中*parameterName*是设定的选项名，*parameterValue*是给*parameterName*指定选项所赋的值。这些参数中惟一的数值参数是收敛容差，其设定方法如下：

⊖ 键入help fzero能够得到所有可能的输入、输出列表。

```
>> options = optimset('TolX',delta)
```

其中*delta*是个正数，一个标量值，它的默认值为eps = 2.2×10^{-16}。

参数'Display'控制fzero迭代过程中的打印信息量，它可设为'iter'、'off'或'final'。设为'final'时（默认值），只打印出最初的有根区间和最终结果；设为'iter'时，打印求解过程中每一个主要步骤的结果；设为'off'时，函数fzero在执行时不打印任何信息。

275

下列语句使用optimset和fzero函数求解sin(x)的零点并打印出每一步的迭代结果：

```
>> options = optimset('Display','iter');
>> r = fzero('sin',[pi/4 3*pi/2],options)

 Func-count       x            f(x)         Procedure
     1        0.785398       0.707107        initial
     2         4.71239             -1        initial
     3         2.41201       0.666558        interpolation
     4          3.5622      -0.408315        bisection
     5         3.12527      0.0163175        interpolation
     6         3.14206   -0.000471667        interpolation
     7         3.14159   2.03284e-008        interpolation
     8         3.14159  -7.65714e-016        interpolation
     9         3.14159   5.66554e-016        interpolation
Zero found in the interval: [0.7854, 4.7124].
r =
    3.1416
```

注意到函数optimset的输出结果可以赋给任意MATLAB变量，也就是说，它不一定要赋给名为options的变量。函数fzero可以在参数列表中调用optimset，这样就根本不需要显式创建options变量。重复前面的计算，不打印输出结果，可以用下列语句：

```
>> r = fzero('sin',[pi/4 3*pi/2],optimset('Display','off'))
r =
    3.1416
```

参数'Display'和'TolX'可同时设定。例如，设一个较大的收敛容差，再次对前面的方程求根，在命令行输入：

```
>> options = optimset('Display','iter','TolX',0.05);
>> r = fzero('sin',[pi/4 3*pi/2],options)

 Func-count       x            f(x)         Procedure
     1        0.785398       0.707107        initial
     2         4.71239             -1        initial
     3         2.41201       0.666558        interpolation
     4          3.5622      -0.408315        bisection
     5         3.12527      0.0163175        interpolation
Zero found in the interval: [0.7854, 4.7124].
r =
    3.1253
```

276

这五个步骤和前面带默认参数的五个步骤完全一样，改变收敛容差只会导致在计算结束时的不同，但是它们收敛的过程是一样的。

对于通用的求根问题，读者最好使用fzero函数，不建议使用本章前面介绍的迭代方法（如区间划分法、牛顿法和割线法）。若函数的函数值及其一阶导数值易于计算，并且是良性

的，牛顿法就比函数fzero收敛速度快。对于简单的函数，除非进行大量重复的求根运算，否则计算时间上的差距可以忽略。

例6.11　用fzero求解野餐桌设计问题

在本例中，用fzero函数求解方程（6-2）中定义桌腿尺寸的角度。基本解法只需要对例6.9中的代码稍作修改即可得到计算结果。另外，这里还示范了函数fzero的一个特性，即避免使用全局变量。关于全局变量的使用请参见例6.9。

要指定野餐桌腿的尺寸，需要求出满足下列方程的θ值

$$f(\theta)=w\sin\theta \quad h\cos\theta-b=0$$

其中w、h和b分别是桌子的宽度、高度以及桌腿型材的宽度（参照图6-3）。函数fzero就是要针对给定的w、h和b值求出θ的值。为使计算灵活，参数w、h和b的值没有硬编码到定义函数$f(\theta)$的m文件中，而是通过fzero把这些参数值传给m文件函数$f(\theta)$。

和前面提到的一样，函数fzero的调用形式为

```
fzero(fun,x0,options,arg1,arg2,...)
```

它把arg1,arg2,...传给由fun参数指定的m文件，程序清单6-9中的函数tablez和legz就采用了这种方法。主程序tablez调用fzero的语句为:

```
theta = fzero('legz',t0,[],0,w,h,b);
```

或

```
theta = fzero('legz',t0,optimset('Display','off'),w,h,b);
```

上面两种语句形式的选择要根据运行时检测到的不同MATLAB版本来决定。传递参数在参数表的最后，这样它们的值就必须提供给可选控制参数（5.2及更早版本中为tol和trace；5.3及其更新版本中为optimset）。在MATLAB 5.2的语法中，第三个参数为空矩阵[]说明函数fzero使用默认的容差。

277

程序清单6-9　用于设计野餐桌腿的函数tablez和函数legz。内置函数fzero用于求根

```
function tablez(w,h,b,t0)
% tablez  Use fzero to find dimensions of picnic table legs
%
% Synopsis:  tablez(w,b,h)
%            tablez(w,b,h,t0)
%
% Input:     w = width of the table legs
%            h = height of the table legs
%            b = width of the stock used to make the legs
%            t0 = (optional) initial guess at theta (in radians)
%                 Default:  t0 = pi/4
%
% Output:    Print out of table dimensions

if nargin<3
   error('All three dimensions, w, h, b, must be specified');
elseif nargin<4
   t0 = pi/4;        % default initial guess
end
[v,d] = version;
```

```
vnum = str2num(v(1:3));   % vnum is version number in d.d format
if vnum<5.3
  % "old" fzero syntax
  theta = fzero('legz',t0,[],0,w,h,b);    %  w,h,b are passed through to legz
else
  % optimset('Display','off') stops printing *and* avoids warning from parser
  theta = fzero('legz',t0,optimset('Display','off'),w,h,b);
end

% --- Compute other dimensions once theta is known
alpha = pi/2 - theta;     a  = b/tan(alpha);      c = b/tan(theta);
d2 = h/(2*sin(theta));    d1 = d2 - a - c;
fprintf('theta = %6.1f degrees   d1 = %8.3f   d2 = %8.3f\n',theta*180/pi,d1,d2);

function f = legz(theta,w,h,b)
% legz  Evaluate f(theta) for picnic leg geometry.  Use pass-through
%       parameters in fzero to send w, h, and b values to this function
f = w*sin(theta) - h*cos(theta) - b;
```

278

这里传递参数变成了fzero函数的第一个参数所指定m文件的附加输入参数。因此，计算$f(x)$的m文件必须有正确的输入参数个数，以便接收来自fzero的传递参数值。函数legz定义为：

```
function f = legz(theta,w,h,b)
```

运行tablez后，得到和例6.9中函数tablen相同的结果：

```
>> tablez(28,27,3.5);
theta =   49.1 degrees   d1 =    10.782   d2 =    17.855
```

6.7 多项式的根

上面讨论的求根程序也可用于多项式求根，因为关于x的多项式都可以写成$f(x)=0$的形式。但是当所求根有多个解、或根之间距离很近、或出现复数根时，用这些方法就很难求解。这样，若要求解，最好用专门设计的用来处理复杂问题的数值算法来求多项式的根。

图6-17画出了三个二次多项式的图形：

$$f_1(x) = x^2 - 3x + 2,\qquad f_2(x) = x^2 - 10x + 25,\qquad f_3(x) = x^2 - 17x + 72.5$$

$f_1(x)$有两个不同的实根，$f_2(x)$有重实根，而$f_3(x)$只有复数根。

279

图6-17 存在不同实根、重实根和复根的多项式对应的图形

4阶以下的多项式都有显式的求根公式，如例5.3所述，计算机程序求下列二次方程

$$ax^2+bx+c=0$$

的根时，不能使用我们所熟悉的二次方程求根公式

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

而应该分两步计算，这样在数值上会更加可靠：

$$q\equiv-\frac{1}{2}\left[b+\mathrm{sign}(b)\sqrt{b^2-4ac}\right]$$

其中

$$\mathrm{sign}(b)=\begin{cases}1, & b\geqslant 0\\ -1, & 其他\end{cases}$$

那么二次方程的根就是：

$$x_1=\frac{q}{a},\qquad x_2=\frac{c}{q}$$

文献[1、61]给出了三次多项式和四次多项式的显式求根公式。

现在已经开发出了专门对多项式求根的程序。Bairstow方法采用综合除法，它先用原始（实数）多项式除以一个二次项因子，以此来降幂（参照文献[3、38、68]），然后使用二次方程成对地求根。Bairstow法的优点在于它能够通过实数的算术计算得到复根。另外的方法是文献[2、6]中提到的Laguerre算法和文献[41、42]中提到的Jenkins-Traub算法。而下一节将要介绍的内置函数`roots`则使用另一种方法求根。

`roots`函数

内置函数`roots`用于求下列形式的多项式的根：

$$c_1x^n+c_2x^{n-1}+\cdots+c_nx+c_{n+1}=0$$

函数`roots`通过计算多项式伴随矩阵（*companion matrix*）的特征值来求出多项式的根（参见附录A.2.2）。所有的根只需要调用一次`roots`函数就能够全部得到。

280

对于下列形式的四阶多项式

$$c_1\lambda^4+c_2\lambda^3+c_3\lambda^2+c_4\lambda+c_5=0 \tag{6-15}$$

多项式的伴随矩阵为

$$\boldsymbol{A}=\begin{pmatrix}-c_2/c_1 & -c_3/c_1 & -c_4/c_1 & -c_5/c_1\\ 1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\end{pmatrix}$$

$\boldsymbol{A}$的特征值为满足方程

$$\boldsymbol{A}v=\lambda v \tag{6-16}$$

的λ值。方程（6-15）称为方程（6-16）中特征值λ的特征多项式。求特征值的一般程序相当

复杂，可是很有价值。由于求特征值的计算难度大，一般不会将求根问题转化为求特征值的问题。但是，MATLAB在线性代数的计算上有自己的优势，它的内置函数eig能够很快地求得矩阵的特征值。有了这种内在优势，MATLAB就可以轻易地将多项式的求根问题转化为求特征值问题。

下列语句用于求出图6-17画出的三个二次多项式的根：

```
>> roots([1 -3 2])
ans =
     2
     1

>> roots([1 -10 25])
ans =
     5
     5

>> roots([1 -17 72.5])
ans =
   8.5000 + 0.5000i
   8.5000 - 0.5000i
```

例6.12　飘浮的球体

一个实心球体浮在水面上，根据阿基米德原理，球体的浮力和球体排出水的重量相等。令 $V_s=(4/3)\pi r^3$ 为球的体积，V_w 为球体局部浸入水中时排出水的体积。在静态平衡的状态下，球的重量和水的浮力相等 281

$$\rho g V_s=\rho_w g V_w \text{ 或 } sV_s=V_w \tag{6-17}$$

其中 ρ 是球体的密度，g 是重力加速度，ρ_w 是水的密度，$s=\rho/\rho_w$ 是球体材料的比重。

如图6-18所示，当 $0<s<0.5$ 时，球体的中心在水面上方，当 $0.5<s<1$ 时，球体的中心在水面的下方。

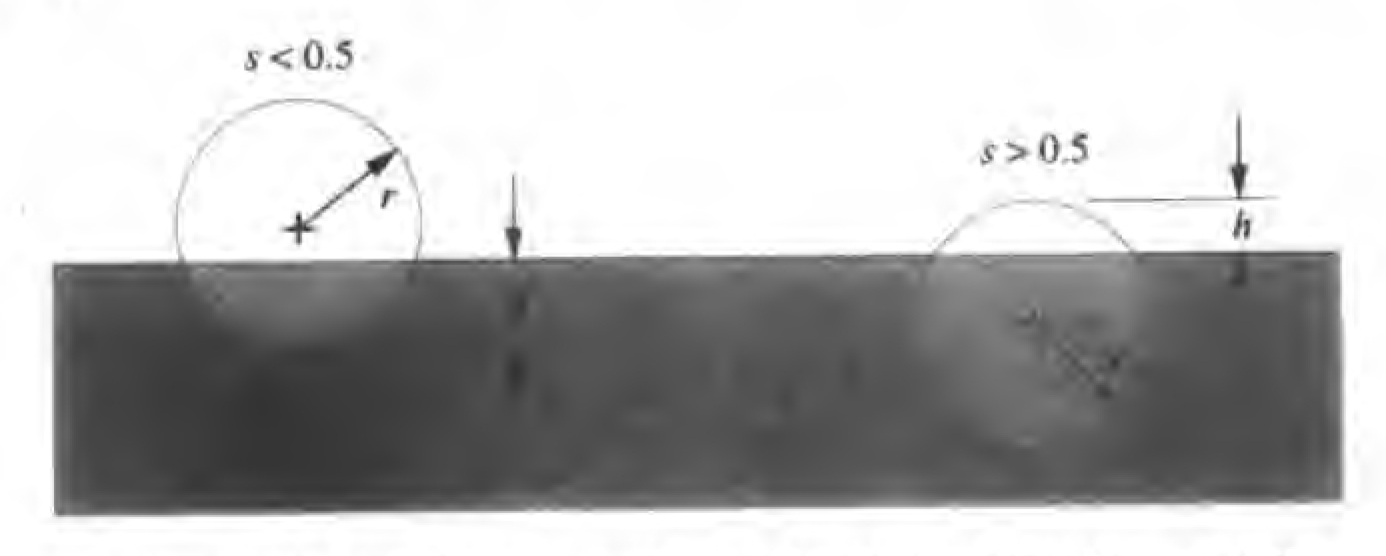


图6-18　浮在水面上的球体。它沉入水面的深度由球体的比重 $s=\rho/\rho_w$ 决定

如图6-18所示，球体高度为 h 部分的体积为

$$V_h=\frac{\pi}{3}(3rh^2-h^3)$$

若 $s<0.5$，那么 $V_w=V_h$，方程（6-17）可整理为

$$h^3 - 3rh^2 + 4sr^3 = 0 \tag{6-18}$$

给定r和s的值，就可以用roots函数求得满足方程（6-18）的h值：

```
>> r = 1;   s = 0.25;            %  (arbitrary) radius and specific gravity
>> p = [1  -3*r  0  4*s*r^3 ];   %  polynomial coefficients
>> h = roots(p)
h =
    2.8794
    0.6527
   -0.5321
```

当$s < 0.5$且$r = 1$时，只有$h = 0.6527$有意义。注意到p的定义中，h项的系数要为零，这一项不能省略。

按照一般的求根步骤，用roots函数求出多项式的根后，要把根代入多项式进行验证，这通过polyval函数可以轻易实现。在上面的语句中，多项式的系数放在向量p中，根的返回值放在向量h中。在猜测根处计算多项式得到

282

```
>> polyval(p,h)
ans =
   1.0e-13 *
    0.1077
    0.0056
   -0.0022
```

和本章介绍的其他求根程序不同的是，这一求根程序无法调整函数roots返回值的容差。

6.8 小结

本章介绍了形如$f(x) = 0$的标量函数的几种求根程序。表6-2列出了本章用到的函数。

表6-2　本章用到的求根程序。小节为N.A. 的函数在NMM工具箱中，但在本章中未列出

函　数	小　节	描　述
bisect	6.3.3	用二分法求$f(x) = 0$的根的通用程序
brackPlot	6.1.3	求出函数的有根区间并画出图形
demoBisect	6.3	用二分法求多项式$x - x^{1/3} - 2$的根
demoNewton	6.4	用牛顿法求$f(x) = x - x^{1/3} - 2$的根
fx3	6.1.3	求函数$f(x) = x - x^{1/3} - 2$的值
fx3n	6.4.2	为牛顿法计算$f(x) = x - x^{1/3} - 2$和df/dx的值
legsn	6.5	计算野餐桌腿问题的$f(\theta)$和$f'(\theta)$，只使用牛顿法
legsnNG	N.A.	计算野餐桌腿问题的$f(\theta)$和$f'(\theta)$，使用牛顿法和tablenNG函数
legz	6.6.1	计算野餐桌腿问题的$f(\theta)$
newton	6.4.2	用牛顿法求$f(x) = 0$的根的通用程序
newtonNG	N.A.	用牛顿法求$f(x) = 0$的根的通用程序，用newtonNG传递附加参数，无需全局变量
tablen	6.5	用牛顿法求野餐桌腿的尺寸
tablenNG	N.A.	用牛顿法求野餐桌腿的尺寸，利用函数newtonNG和legsnNG避免使用全局变量
tablez	6.6.1	用fzero求野餐桌腿的尺寸

283

对于非多项式$f(x)$求根，定点迭代法、二分法、牛顿法和割线法都适用。求出根后，最好

把所求结果代入$f(x)$检验是否得到$f(x) \approx 0$。特别地，区间划分法和二分法不能区分开$f(x) = 0$的根和奇点。通常情况下，当初始猜测值未知时，使用区间划分法能有效地求出包含根的区间。用6.1.3节介绍的函数`brackPlot`能方便地在给定区间画出用户自定义的函数$f(x)$的图形，并在此区间上找出有根区间。

强烈建议使用内置函数`fzero`，它结合了二分法的可靠性特点以及割线法与二次反插值的高效性特点。用`fzero`求$f(x) = 0$的根，用户必须定义一个m文件函数来计算任意x的$f(x)$值。由于函数`brackPlot`需要相同程序作为输入，这样，用户几乎不用再做什么工作就可以画出函数图，用`brackPlot`自动产生猜测值，然后将这些值传给`fzero`函数。

关于x的多项式的$f(x)$函数和一般的$f(x)$不同，多项式求根最有效的程序是根据具体情况专门设计的，本章只介绍了伴随矩阵求多项式根的方法。其中，伴随矩阵的特征多项式就是要求根的多项式。因此，求根问题就可以转换成求特征值问题，而使用高度优化的MATLAB内置函数可以很快求出特征值，函数`roots`将这些复杂的过程编在程序中，并返回多项式的所有实根和复根。

补充读物

大多数数值分析方面的教科书中都涉及到了求根问题。Forsythe等人在文献[24]中用Fortran实现了`fzero`函数，Press等人在文献[6]中用Fortran和C实现了这个函数。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1. （1）证明方程（6-2）的解析解是

$$\cos\theta = \frac{1}{(h^2+w^2)}\left(-bh+\sqrt{b^2h^2+(h^2+w^2)(w^2-b^2)}\right)$$

（提示：对方程两边作平方运算，然后用$\sin^2\theta = 1-\cos^2\theta$代入方程。）

284

2. *（2）函数$f(x) = \sin(x^2) + x^2 - 2x - 0.09$在区间$-1 < x < 3$上有四个解。给定的m文件`fx.m`包含下列代码

```
function f = fx(x)
f = sin(x.^2) + x.^2 - 2*x - 0.09;
```

而语句

```
>> brackPlot('fx',-1,3)
```

只得到两个有根区间。是否由于`brackPlot`或`fx`中有bug，才导致这种结果呢？怎样才能得到四个根？并证明自己的解法。

3. （1）用`brackPlot`函数求下列方程的根，在给定区间内它们分别存在几个根？

(a) $\cos(x) = x$，$2\pi \leqslant x \leqslant 2\pi$

(b) $\cos(5x) = x$，$-2\pi \leqslant x \leqslant 2\pi$

(c) $\tan(x) = 0$，$-2\pi \leqslant x \leqslant 2\pi$

(d) $\sin(1/x) = 0$，$\pi/50 \leqslant x \leqslant \pi/3$

4. （1）将`brackPlot`函数中画图的代码段去掉，编写一个`bracket`函数，用练习3中的方程检验函数`bracket`的正确性。

5. （2）算法6.1中，while循环的测试条件为什么用“while $i<n$”来代替“while $a<b$”？
6. （1）给定函数cos $(x)=x$，推导两个定点迭代公式，当初始猜测值为$x_0=0.5$弧度时，哪个迭代公式收敛？
7. （1）证明例6.4中迭代函数的行为与定点迭代的收敛准则$|g'(x)|<1$一致。
8. （1+）考虑完全充满流体的管道关于摩擦因子的Colebrook方程

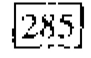
$$\frac{1}{\sqrt{f}}=-2\log_{10}\left(\frac{\varepsilon/D}{3.7}+\frac{2.51}{Re_D\sqrt{f}}\right)$$

其中f是Darcy摩擦因子，ε/D为管道材料的相对粗糙度，Re_D是基于管道直径D的Reynolds数。最简单的管道流量分析中，ε/D和Re_D的值已知，f为待求值。导出下列两个可选的定点迭代公式：

$$f_{new}=\left[2\log_{10}\left(\frac{\varepsilon/D}{3.7}+\frac{2.51}{Re_D\sqrt{f_{old}}}\right)\right]^{-2}$$

以及

$$f_{new}=\left[\frac{Re_D}{2.51}\left(10^{-1/(2\sqrt{f_{old}})}-\frac{\varepsilon/D}{3.7}\right)\right]^{-2}$$



编写两个m文件函数实现上面两个迭代公式，求出当$\varepsilon/D=0.02$、$Re=10^5$且初始猜测值$f=0.01$时的f值，并给出每种公式收敛前的迭代次数。
9. （1）用函数bisect求出练习3中每个方程的根。怎样判断是否求出了所有根？
10. （1+）求方程$f(x)=(2/x-x)\cos x+0.0119$在区间[0.5，5]上的根？
11. *（1+）用函数bisect求出当$\varepsilon/D=0.02$，$Re=10^5$时Colebrook方程（参照练习8）的根。注意，不能修改bisect.m文件，要求读者编写适当的m文件函数计算Colebrook方程。
12. （2）编写函数autoRoot，要求包含6.3.3节末的代码完成自动查找和改进猜测根。输入为计算$f(x)$的m文件名和起始有根区间的两个端点值（xmin和xmax）。另外，此函数还要提供可选的第四个输入变量nx，用于指定brackPlot寻找根的子区间个数。在区间$-1\leqslant x\leqslant 3$上用autoRoot求函数$f(x)=\sin(x^2)+x^2-2x-0.09$的四个根，以检验此函数。
13. *（1）导出例6.4和例6.5的$g_3(x)$函数。（提示：牛顿法的定点公式是什么？）
14. （1）用牛顿法导出cos $(x)=x$的迭代求根公式。初始猜测值为$x=5$弧度，计算经过5次迭代后根的估计值。对初始猜测值$x_0=\pi$、$x_0=3\pi/2$、$x_0=2\pi$分别求出当$|f(x)|<5\times10^{-10}$时的迭代次数。
15. （1）用牛顿法迭代公式导出方程（3-3）中计算$\sqrt{x}$的公式（提示：定义$f(r)=r^2-x$，其中x是要求平方根的数值）。
16. （2）用函数newton求出方程$f(x)=\sin(x^2)+x^2-2x-0.09$在区间$-1\leqslant x\leqslant 3$上的四个根，并给出求每个根所用的初始猜测值。
17. *（2+）K.Wark和D.E.Richards（*Thermodynamics*,6th ed.,1999,McGraw-Hill,Boston, Example14-2,pp.768–769）计算在一个大气压下一氧化碳和氧气的混合气体的合成平衡。要解出最终的合成，需要解下面关于x的方程：

$$3.06=\frac{(1-x)(3+x)^{1/2}}{x(1+x)^{1/2}}$$

给出上面方程求根的定点迭代公式。用MATLAB函数实现此公式并求出x的值。要求你所推导的公式收敛。

18.（2）利用函数`bisect`编写m文件求解练习17中的方程。

19.（2）利用函数`newton`编写m文件求解练习17中的方程。当初始猜测值为$x=0.3$和$x=1$时，分别给出迭代的情况。

20.（1+）用牛顿法编写一个m文件解决输入值w、h、b为任意值时的野餐桌设计问题。 [286]

21.（1+）David Peters(*SIAM Review*,vol.39,no.1,pp.118-122,March 1997)得到震荡阻尼系统设计时获得最佳阻尼系数的方程如下，当给物体施加作用力时，要求传送的力最小：

$$\cos\left[4\zeta\sqrt{1-\zeta^2}\right]=-1+8\zeta^2-8\zeta^4$$

编写m文件求出满足上式的ζ值。

22.（1+）散热片用于加速物体的冷却，散热片产品有不同的形状和大小，但在批量生产时，要将其形状优化。典型的优化就是要以最低的成本得到最大的热能效率。成本通常以材料的多少来判定。对圆柱形散热片的优化就是要找出满足$10\beta=3\sinh(2\beta)$的β值（参值A.D.Kraus和A.Bar-Cohen.*Design and Analysis of Heat Sinks*,1995, Wiley, NY,p.250）。找出使圆柱形散热片最优化的大于零的β值。

23.（2+）通常将散热器附着在电子设备上以提高冷却效率，从而降低电子设备的温度。通常使用称为针式散热棒阵列的散热器，如插图所示。给定阵列的尺寸L、H和W，要求计算出散热棒的大小及最优间距。Adrian Bejan（"Geometric optimization of cooling techniques", pp.1-45 in *Air Cooling Technology for Electronic Equipment*, S.J.Kim and J.S. Woo, eds., 1996, CRC Press）提出了如下优化间距（S_{opt}）的公式：

$$\frac{S_{opt}}{D}\frac{2+S_{opt}/D}{(1+S_{opt}/D)^{2/3}}=2.75\left(\frac{H}{D}\right)^{1/3}Ra^{-1/4}$$

其中D是散热棒的直径，Ra是Rayleigh数，它是起散热作用的自然对流强度的一种无量纲指标。要求编写m文件函数对给定D、H和Ra计算最优间距，并画出区间$300\leqslant Ra\leqslant 10000$上$H/D=5$、10、15、20的$S_{opt}$图形。



[287]

24.*（2+）修改`newton`函数（新函数可命名为`newtonb`），要求它的输入变量为一个有根区间（代替原单个初始猜测），用二分法由区间端点得出近似值x_0，作为牛顿迭代的

初始猜测值，并仿照程序清单6-4中的bisect函数分别求出x和$f(x)$的相对容差。

25. (3) 要求用一步牛顿法编写一个工具程序newtBrack，检验猜测根是否在有根区间中。此函数定义应有如下形式：

```
function [ok,xnew] = newtBrack(a,b,x,f,fprime)
```

其中a和b是区间的端点值；x是根的当前猜测值；f和fprime是x处的$f(x)$和$f'(x)$值；xnewt是用牛顿法得到的新近似值。根据xnewt是否在$a \leqslant x \leqslant b$区间内，ok的值取1或0。要求在上个练习中的newtonb函数中加入对newtBrack的调用。修改后的newtonb函数应该能进行正常的牛顿迭代，且在ok的值为“false”时打印警告信息。这段代码是一个更大的编程项目中的一部分，此项目在下面的练习中完成。

26. (3+) 使用前面两个习题中的newtBrack和newtonb函数编写函数newtSafe，要求仅当经过一步牛顿迭代就可以在起始有根区间内得到根时，才进行这步牛顿迭代。若牛顿法得到的根在起始有根区间外，就打印警告信息并换用二分法迭代（注意可以将函数newtBrack分两行写，这样能够把它合并到函数newtSafe中）。用所编函数newtSafe求解习题17中$f(x)$在起始有根区间[0.3，1.2]上的x，以此来检验newtSafe函数的正确性。

27. * (3-) 用算法6.5和方程（6-13）实现割线法。要求用例6.10测试所编程序的正确性。进行10次迭代后将会出现什么情况？用下列公式替换方程（6-13）

$$x_{k+1} = x_k - f(x_k)\left[\frac{(x_k - x_{k-1})}{f(x_k) - f(x_{k-1}) + \varepsilon}\right]$$

其中ε和ε_m一样，是个很小的数。这样结果将如何改变？为什么？

28. (3-) 用算法6.5和方程（6-14）实现割线法。要求用例6.10测试所编程序的正确性。进行10次迭代后将会出现什么情况？用下列公式替换方程（6-13）

$$x_{k+1} = \frac{f(x_k)x_{k-1} - f(x_{k-1})x_k}{f(x_k) - f(x_{k-1}) + \varepsilon}$$

其中ε和ε_m一样，是个很小的数。将得到的结果与练习27的结果相比较，结论如何？哪个公式的数值特性更好？

288

29. (3) 用MATLAB实现割线法，函数的序言为

```
function r = secant(fun,xb,xtol,ftol,verbose)
% secant  Secant method for finding roots of scalar f(x) = 0
%
% Synopsis:  r = secant(fun,xb)
%            r = secant(fun,xb,xtol)
%            r = secant(fun,xb,xtol,ftol)
%            r = secant(fun,xb,xtol,ftol,verbose)
%
% Input: fun     = (string) name of function for which roots are sought
%        xb      = vector of bracket endpoints. xleft = xb(1), xright = xb(2)
%        xtol    = (optional) relative x tolerance.    Default:  xtol=5*eps
%        ftol    = (optional) relative f(x) tolerance. Default:  ftol=5*eps
%        verbose = (optional) print switch. Default: verbose=0, no printing
%
% Output:  r = the root of the function
```

用算法6.5和修改练习27中的代码完成这个函数。结合函数fx3求解方程（6-3），以证明你的函数有效。函数fx3和secant函数的(shell)命令解释程序在NMM工具箱的rootfind目录下。

30. （2+）函数fzero（MATLAB 第五版）的第81行的代码用语句

```
if (fa > 0) ==(fb > 0)
```

来判定函数在区间$a < x < b$中是否改变符号。为什么不用if　fa*fb<0？又为什么不用sign(fa) == sign(fb)呢（提示：sign(0)是多少？）？

31. （1+）编写一个m文件函数，其输入为ε/D和Re，输出为满足Colebrook方程的f值（参照练习8）。编程时使用fzero，不要使用全局变量。根据下列公式可以得到起始猜测根：

$$f=\left\{1.8\log_{10}\left[\frac{6.9}{Re}+\left(\frac{\varepsilon/D}{3.7}\right)^{1.11}\right]\right\}^{-2}$$

它是满足Colebrook方程的f的一个近似。对下列ε/D和Re，分别求出f的值：

(a) $\varepsilon/D=0$，$Re=5000$

(b) $\varepsilon/D=0.0001$，$Re=3\times10^5$

(c) $\varepsilon/D=0.0001$，$Re=5\times10^7$

(d) $\varepsilon/D=0.03$，$Re=1\times10^4$

(e) $\varepsilon/D=0.01$，$Re=3\times10^5$

289

32. （1+）在Re很小时，管道内流体的摩擦因子为

$$f=\frac{64}{Re}\quad 对\quad Re\leqslant 2000,\quad 任意\quad \varepsilon/D$$

把上一道练习的结果进行扩展，要求能够计算Re很小（也称薄片结构（laminar））的情况下流体的摩擦因子。分别计算下列情况的结果：

(a) $Re=1000$

(b) $Re=2000$

(c) $Re=2001$

(d) $\varepsilon/D=0.001$，$Re=2001$

33. *（3－）编写m文件函数计算半径为r，比重为s的浮球体浸入水的深度h（参见例6.12）。输入为r和s，输出为h。当$s<0.5$时，计算h的值。$s>0.5$的情况在下面的练习中计算。当$s>0.5$时，打印错误信息并终止程序（使用内置函数error）。要求所编函数能够从内置函数roots的返回值中选出正确的根。

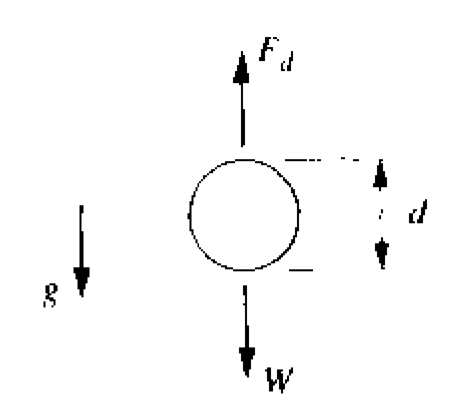
34. （3－）当$s\geqslant0.5$时，导出类似方程（6-18）的方程。并用此方程扩展上一练习得到的函数，使其能处理$0<s<1$范围的所有值。通过检查输入的s值，选择合适的公式来定义关于h的三次多项式。为了简单起见，精确地测试$s=0$、$s=0.5$和$s=1$时的结果。$s=0.5$时的h值可通过物理推理得到，不需要解方程。按照防错性程序设计机制，程序也要测试输入值$s<0$和$s>1$的情况。

35. （2－）用上面练习编写的m文件函数画出h-s的函数关系图，其中s = linspace(0.0001,0.9999,50)。

36. （2+）方程$x\tan(x)=c$在区间$0\leqslant x\leqslant2\pi$中有两个根，其中$c$为一个已知参数。画出这两个根随$c$在$0.1\leqslant c\leqslant10$范围内变化的函数图。当$c=0$时，两个根为多少？$c=0$会给求

根程序带来麻烦吗？

37. (3+) 下面的插图描述了一个直径为d的球体在空气中下落的情况：



290

球体被释放后会一直加速，直到终极速度。当阻力（drag force）F_d和球体重力$\omega = mg$平衡时，达到终极速度：

$$F_d = mg \tag{6-19}$$

m为球体的质量，g为重力加速度。阻力F_d可由下式计算

$$F_d = c_d \frac{1}{2}\rho v^2 A \tag{6-20}$$

其中c_d是经验阻力系数，ρ是流体的密度（此时是空气），v是球体的速度，$A = (\pi d^2)/4$是球体的表面积。F.M White(*Viscous Fluid Flow*,2d ed.,1991,McGraw-Hill,p.182)给出了球体空气动力学阻力实验数据的曲线拟合：

$$c_d = \frac{24}{Re} + \frac{6}{1+\sqrt{Re}} + 0.4, \quad 0 < Re < 2\times 10^5 \tag{6-21}$$

$Re = (\rho v s)/\mu$是Reynolds数，μ是流体（空气）的动态粘度。把方程（6-20）和（6-21）代入方程（6-19）可以得到终极速度和重量之间的关系式。c_d仍然用关于Re的表达式来表示（即使你需要把v代入Re的表达式中来计算公式的值）。将得到的公式变换成$f(x) = 0$的形式。

以函数`tablez`为模型（参照例6.11），编写两个m文件求解任意球体在空气中下落的终极速度。一个m文件接收用户从命令行输入的m值和d值，另一个m文件为求根程序计算$f(x)$的值。m、d和μ的值怎样传给$f(x)$的m文件？不要使用全局变量。用所编m文件函数给出下列情况的终极速度：

(a) m = 2 gm，d = 2 cm

(b) m = 2 kg，d = 15 cm

(c) m = 200 kg，d = 1 m

方程（6-21）对任何m值和d值都有效吗？

38. (3+) 练习37的方程（6-21）仅在Re的一个有限的范围内有效。当$Re \sim 10^5$时，c_d的值会在某个点处急剧减小，此点就称为阻力临界点。阻力临界点的Re值随球体的粗糙程度不同而不同，当球体比较粗糙时，Re的值比较小。高尔夫球比较粗糙，所以它会在低速情况下出现阻力临界点，从而减小空气动力学阻力。

291
292

NMM工具箱的data目录下的`sphereCd.dat`数据文件中包含了光滑球体在$2\times 10^4 < Re < 3.99\times 10^6$范围内的$c_d = f(Re)$数据。这些数据来自R.D. Metha，“Aerodynamics

of Sport Balls" in *Annual Review of Fluid Mechanics*, M.Van Dyke(ed.),Vol.17,pp.151-189,1985,Annual Reviews, Inc。对给定的Re值编写一个m文件函数（取名为sphereCd），返回合适的C_d值。当$0 \leqslant Re \leqslant 2 \times 10^4$时，用方程（6-21）计算；当$2 \times 10^4 \leqslant Re \leqslant 3.99 \times 10^6$时，对sphereCd.dat中的数据进行线性插值（可以使用内置函数interp1）；当$Re > 3.99 \times 10^6$时，假设c_d为它在$Re = 3.99 \times 10^6$处的常量。

用所编sphereCd函数代替方程（6-21）重新计算习题37的终极速度。用函数sphereCd得到的三种球体的终极速度与仅使用方程（6-21）得到的有何不同?

第7章 线性代数回顾

线性代数是一门用来研究对矩阵和向量系统的操作的学科。它的应用面很广，其中的两种最普遍的应用是：线性方程的求解和矩阵特征值的计算。其他的应用包括：计算机图形学（图形处理）、动态系统时间积分、统计分析、曲线拟合以及优化等。线性代数在矩阵和向量的操作上有严格的规则，本章就介绍这些规则。第8章将介绍方程组的求解方法。

图7-1总结了本章讲述的内容。前三节介绍向量、矩阵以及它们的数学性质，请读者认真阅读，为学习第8章做好准备。其中，特殊矩阵一节读者只要浏览一下，当需要时再仔细阅读它。

本章重点介绍线性代数的基础知识，MATLAB中的函数和程序不作为重点。只有读者完全理解矩阵和向量操作的数学规则后，才能够自如地用MATLAB进行操作。同时，读者要不断地使用MATLAB进行练习，才能验证自己的理解应用在实际问题中是否正确。“MATLAB”的名字是“Matrix Laboratory”的缩写。MATLAB是一个学习线性代数的良好环境，在它的命令窗口中就能很容易验证你对矩阵操作的理解是否正确。本章提供的素材就是为这种操作实验准备的，建议读者重写或修改文中的MATLAB语句并验证其正确性。通过实验，读者就能够对线性代数有更直观的理解。

293

本章主题
1. **向量**
定义了向量及对向量的数学操作
2. **矩阵**
定义了矩阵及对矩阵的数学操作。矩阵—向量和矩阵—矩阵积的计算被理解为对行和列的操作
3. **向量和矩阵的数学性质**
简要总结了向量空间的理论知识，定义了矩阵的范围和秩
4. **特殊矩阵**
有些“特殊”的矩阵有特殊的性质，它们能够简化计算

图7-1 第7章的主题

7.1 向量

向量是按行或按列排列的实数或复数的有序集。向量中的数称为元素。书写向量时，要用大圆括号或方括号把元素括起来。包含m个元素的列向量x的书写格式如下：

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

包含n个元素的行向量y的书写格式如下：

$$y = [y_1, \ y_2, \ \dots, \ y_n]$$

有时，行向量的每个元素之间不写逗号，如$y = [y_1 \ y_2 \ \cdots \ y_n]$。用小写的罗马字母表示整个向量，给向量名添加下标就能够引用向量中的元素。例如，向量x的第三个元素为x_3，第i个元

294

素为x_i。

向量通常用来表示包含方向信息的物理量，如力、加速度、速度和位移。由于物理空间是三维的，所以这些物理量向量的元素不会超过三个。在方程中，通常给这些物理量添加箭头号表明它们是向量。如，牛顿运动定律可写为

$$\vec{F} = m\vec{a}$$

这里$\vec{F}$表示外加作用力向量，m（标量）是物体的质量，$\vec{a}$是加速度向量。由于大写和小写字母经常会混用，有的小写字母是标量标识，所以使用箭头更有助于明确区分向量和标量。

当求解线性方程组时，使用箭头就会很繁琐，所以就使用表7-1中总结的约定符号，本书通篇将按照这一约定使用各种符号。当小写罗马字母用于表示工程中的标量时，将不按照这一约定使用符号。例如，对于上面的方程，m表示物体的质量，是个标量，而不是向量。

表7-1 标量、向量和矩阵的印刷约定

变量类型	印刷约定	例子
标量	小写希腊字母	σ，α，β
向量	小写罗马字母	$\boldsymbol{u}$、$\boldsymbol{v}$、$\boldsymbol{x}$，$\boldsymbol{y}$、$\boldsymbol{b}$
矩阵	大写罗马字母	$\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$

295

表7-1中定义的简洁约定有助于读者对线性代数的熟练应用。当见到一个不熟悉的表达式时，读者有必要考虑的是："根据上下文的关系判断这些数可能是哪种数据类型？"因此，对于下列表达式

$$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$$

读者能够推出：$\boldsymbol{A}$是一个矩阵（它用大写字母表示），$\boldsymbol{x}$是一个列向量（它用小写字母表示，所以是向量，只有列向量能够右乘一个矩阵，所以是列向量），$\boldsymbol{y}$是一个列向量（它用小写字母表示，所以是向量，向量与矩阵右乘的结果总是列向量）。而且，通过观察，要使$\boldsymbol{Ax}$有效，那么$\boldsymbol{A}$的列数等于$\boldsymbol{x}$的行数。

7.1.1 向量操作

本节介绍了向量的加法、减法和乘法，并定义了向量的内积，引出了正交向量概念以及用范数来度量向量的大小。

向量加法与减法 向量的加法、减法是指两个具有相同元素个数的向量其对应的元素进行加或减操作。计算时可以使用符号和下标来表示。若$\boldsymbol{c}$是向量$\boldsymbol{a}$和$\boldsymbol{b}$的和，可以用符号表示为$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b}$，用下标表示为$c_i = a_i + b_i$。这两种表示方法之间的关系可用⇔来指定。那么，两个n维向量的加减操作可以表示为

$$\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{b} \Leftrightarrow c_i = a_i + b_i \qquad i = 1, \ldots, n$$

$$\boldsymbol{d} = \boldsymbol{a} - \boldsymbol{b} \Leftrightarrow d_i = a_i - b_i \qquad i = 1, \ldots, n$$

这里⇔读作"等价于"。两个向量的行数或列数相等时才能进行加减操作。因此，上面的式子中$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$和$\boldsymbol{d}$必须都为包含n个元素的行向量或包含n个元素的列向量。

下面是MATLAB中向量进行加减操作的例子：

```
>> a = 0:2:10;    b = 0:10:50;
>> c = a + b
```

```
c =
     0    12    24    36    48    60

>> d = a - b
d =
     0    -8   -16   -24   -32   -40
```

与标量相乘 一个标量乘一个向量就是把这个标量乘以向量中的每个元素。假设a和b是包含n个元素的行向量或列向量，σ为一个标量。那么

$$\boldsymbol{b}=\sigma\boldsymbol{a} \Leftrightarrow b_i=\sigma a_i \quad i=1,\cdots,n$$

在MATLAB中，标量表示符和向量表示符比较相似，在进行向量和标量相乘时要参照上下文，看下列语句：

296

```
>> u = 0:2:10;      % a vector
>> s = 5;           % a scalar
>> t = s*u
t =
     0    10    20    30    40    50
```

把上面的规则扩展到标量除法的示例如下所示：

```
>> u = 0:2:10;
>> u/4
ans =
     0    0.5000    1.0000    1.5000    2.0000    2.5000
```

向量转置 对于一个行向量和一个列向量，即使它们的元素完全相同，但它们也是不等价的。通过使用转置算符（用上标"T"标识），能够把行向量转换成列向量。例如，假设u是行向量

$$\boldsymbol{u}=[1,2,3], \text{那么} \ \boldsymbol{u}'=\begin{bmatrix}1\\2\\3\end{bmatrix}$$

同样，若v为列向量

$$\boldsymbol{v}=\begin{bmatrix}4\\5\\6\end{bmatrix}, \text{那么} \ \boldsymbol{v}'=[4,5,6].$$

通常，书写时为了节省空间，把列向量写成行向量的转置。例如，$\boldsymbol{a}=[a_1, a_2, \ldots, a_m]^T$是个列向量，在MATLAB中，在向量或向量表达式的末尾附加一个单引号来标识向量的转置运算符，如：

```
>> u = [1 2 3];
>> u'
ans =
     1
     2
     3
```

向量的线性组合 用标量乘以向量能够改变向量中每个元素的大小，向量相加把两个维数相同的向量中对应元素相加。同时用这两个操作能够对两个或多个向量进行线性组合（linear combination）。例如，给定标量α和β，向量u和v，用它们的线性组合形成第三个向量w，$\alpha\boldsymbol{u}+\beta\boldsymbol{v}=\boldsymbol{w}$。

297

$$\alpha\begin{bmatrix}u_1\\u_2\\\vdots\\u_n\end{bmatrix}+\beta\begin{bmatrix}v_1\\v_2\\\vdots\\v_n\end{bmatrix}=\begin{bmatrix}\alpha u_1+\beta v_1\\\alpha u_2+\beta v_2\\\vdots\\\alpha u_n+\beta v_n\end{bmatrix}=\begin{bmatrix}w_1\\w_2\\\vdots\\w_n\end{bmatrix}$$

向量加是线性组合的特例，它对应的标量乘数都为1。任意相同“形状”的向量都能够线性组合，但是，行向量和列向量不能线性组合，包含两个元素的行向量也不能与包含四个元素的行向量线性组合。

作为具体的示例，向量$w=[4,\ 2]^T$可由下列线性组合得到：

$$w=\begin{bmatrix}4\\2\end{bmatrix}=4\begin{bmatrix}1\\0\end{bmatrix}+2\begin{bmatrix}0\\1\end{bmatrix}$$

用来构造向量w的其他等效方法如下所示：

$$w=6\begin{bmatrix}1\\0\end{bmatrix}-2\begin{bmatrix}1\\-1\end{bmatrix},\qquad w=\begin{bmatrix}2\\4\end{bmatrix}-2\begin{bmatrix}-1\\1\end{bmatrix}$$

和

$$w=2\begin{bmatrix}4\\2\end{bmatrix}-4\begin{bmatrix}1\\0\end{bmatrix}-2\begin{bmatrix}0\\1\end{bmatrix}$$

在图7-2中用箭头画出了这些向量，箭尾都在原点$(x,\ y)=(0,\ 0)$，箭头的尖头在$(x,\ y)$处，x、y坐标值分别对应向量中的第一个元素和第二个元素。

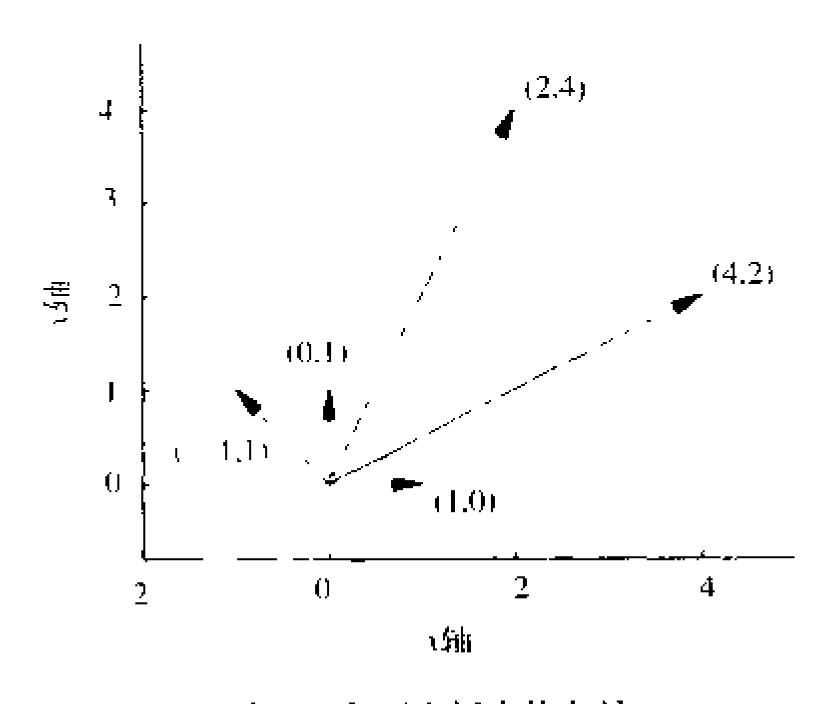


图7-2　在$\mathbf{R}^2$空间中的向量

298

向量的内积　向量的内积（*inner product*）也称为向量的点积（*dot product*），要求进行内积的两个向量必须包含相同个数的元素。内积的结果是一个标量而不是向量。假设x是n维行向量，y是n维列向量，σ是一个标量。x和y内积的结果为

$$\sigma=x\cdot y\iff\sigma=\sum_{i}^{n}x_iy_i\qquad\begin{matrix}x\text{是一个行向量}\\y\text{是一个列向量}\end{matrix}$$

用一个点表示两个向量的内积符合人们的习惯而且很方便，但是，我们需要一个更准确的符号来表示内积。根据线性代数的规则，两个行向量、两个列向量都不能进行内积。只有

下列情况下，两个向量才能够进行内积：行向量在左，列向量在右。假设**x**是四维的行向量，**y**是四维的列向量，那么

$$[x_1 \quad x_2 \quad x_3 \quad x_4]\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4$$

若两个列向量内积，可以把左边的向量先转置，然后内积。因此，对于两个n维的列向量**u**和**v**，它们的内积是

$$\sigma = \mathbf{u}^T\mathbf{v} \quad \Leftrightarrow \quad \sigma = \sum_{i=1}^{n} u_i v_i \qquad \mathbf{u}\text{和}\mathbf{v}\text{是列向量}$$

若**s**和**t**是n维的行向量，用符号表示为$\sigma = \mathbf{s}\mathbf{t}^T$。内积有可交换性，因此（对两个列向量来说）

$$\mathbf{u}^T\mathbf{v} = \mathbf{v}^T\mathbf{u}$$

总之，"点号"本身已经说明向量的大小相等而且进行了转置操作。在手工计算或用Fortran和C编程计算时，内积中使用的转置只是数学形式上不同。计算时，只是简单地把对应元素的积相加来求结果。在MATLAB中，行向量和列向量是完全不同的。

在MATLAB中，符号"*"用于计算两个向量的内积。若这两个向量都是行向量或都是列向量，那么乘之前需要先转置其中的一个向量。两个不匹配的向量相乘会导致错误。看下面 299 的例子：

```
>> u = (0:3)';
>> v = (3:-1:0)',
>> s = u*v
??? Error using ==> *
Inner matrix dimensions must agree.

>> s = u'*v
s =
     4
>> t = v'*u
t =
     4
```

由于转置算术符号比较小，容易被忽略，但是，当转置算术符号被忽略时，MATLAB会提示用户。

使用dot命令计算两个向量的内积时，系统不会考虑它们是否是匹配的行向量或列向量。有兴趣的读者可以使用"type dot"命令获得更多的信息。

例7.1　发票计算中的内积

内积用于那些看似和线性代数无关的计算中。思考计算下列发票中的总成本所要进行的操作：

项	数量	说明	单位成本	单项成本
1	2	成盒铅笔	1.75	3.50
2	2	成盒钢笔	2.25	4.50
3	3	笔记本	1.50	4.50

（续）

4	2	橡皮擦	0.25	0.50
5	1	三孔活页夹	2.75	2.75
			总计	15.75

下列语句说明了无论由每个项目成本项之和还是由数量向量与单价向量的内积都能够得到总的成本：

```
>> quantity = [2 2 3 2 1]';
>> unitCost = [1.75 2.25 1.50 0.25 2.75]';
>> itemCost = quantity.*unitCost;      % array op., not inner product
>> total = quantity'*unitCost          % inner product
total =

   15.7500

>> check = sum(itemCost) - total
check =
     0
```

300

根据第2章关于数组运算符（*array operator*）（.*）的介绍我们知道它是把两个向量对应元素相乘并得到一个新的向量。

向量的外积 向量-向量相乘的另一种类型称为外积（*outer product*），它的结果是一个矩阵而不是一个标量。它不像点积，不能用算符标识。两个列向量$\boldsymbol{u}$和$\boldsymbol{v}$的外积可写为

$$\boldsymbol{A}=\boldsymbol{u}\boldsymbol{v}^{\prime} \quad \Leftrightarrow \quad a_{i,j}=u_i v_j \tag{7-1}$$

两个4×1列向量的外积用符号的形式可表示为

$$\boldsymbol{u}\boldsymbol{v}^{\prime}=\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}\begin{bmatrix} v_1 & v_2 & v_3 & v_4 \end{bmatrix}=\begin{bmatrix} u_1v_1 & u_1v_2 & u_1v_3 & u_1v_4 \\ u_2v_1 & u_2v_2 & u_2v_3 & u_2v_4 \\ u_3v_1 & u_3v_2 & u_3v_3 & u_3v_4 \\ u_4v_1 & u_4v_2 & u_4v_3 & u_4v_4 \end{bmatrix} \tag{7-2}$$

从方程（7-1）和（7-2）看出，矩阵$\boldsymbol{A}$各行中的元素只是在原来向量的基础上多出了几倍（标量相乘得来的），同样，矩阵$\boldsymbol{A}$的各列也是这样。因此，外积得到的矩阵是奇异的（*singular*），特别地，外积运算所得矩阵秩为1。关于秩的概念在7.3.4节中有介绍。

下面是MATLAB进行外积计算的例子：

```
>> u = (0:4)';      % column vector u = [0 1 2 3 4]'
>> v = (4:-1:0)';   % column vector v = [4 3 2 1 0]'
>> A = u*v'         % outer product creates a matrix
A =
     0     0     0     0     0
     4     3     2     1     0
     8     6     4     2     0
    12     9     6     3     0
    16    12     8     4     0
```

外积遵循矩阵乘的规则（参见7.2.2节），即要求左边的向量为$m \times 1$维向量，右边的向量为$1 \times n$维向量。

7.1.2 向量的范数

包含两个元素的向量是二维向量，包含n个元素的向量是n维向量。在许多情况下，我们需要比较相同维数的两个向量的大小。本书所指向量的大小是指其长度的大小而不是维数。所以，单位向量（不论几维）它的大小是1，零向量（所有元素是0）的大小是0。我们可以说：单位向量肯定比零向量大。对任意的n维向量，我们怎么比较它们的大小？

301

首先，看看我们熟悉的关于两个标量绝对值大小的比较。当满足下列式子时，α的绝对值大于β的绝对值

$$|\alpha|>|\beta|$$

这里，α和β都是标量。向量范数（*vector norm*）用于计算和比较向量的大小，它和标量的绝对值相似。与用绝对值运算符来衡量标量（无论正数或负数）大小一样，用范数来衡量向量的大小而不管向量中元素的符号。对于向量$\boldsymbol{x}$和$\boldsymbol{y}$，当满足下列式子时，$\boldsymbol{x}$向量的大小大于$\boldsymbol{y}$向量的大小

$$\|\boldsymbol{x}\| > \|\boldsymbol{y}\|$$

这里$\|\boldsymbol{x}\|$读作"$\boldsymbol{x}$的范数"。

比较二维和三维向量直观的方式是比较他们的几何长度。参考图7-3二维空间中的向量图。假设ℓ是这些向量的长度：

$$\ell_a=\sqrt{4^2+2^2}=\sqrt{20}\quad \ell_b=\sqrt{2^2+4^2}=\sqrt{20}\quad \ell_c=\sqrt{2^2+1^2}=\sqrt{5}$$

将长度的定义扩展到n维向量得到：对于n维向量的长度用L_2范数或欧几里得（*Euclidean*）范数计算，如下所示：

$$\|\boldsymbol{x}\|_2=\left(x_1^2+x_2^2+\cdots+x_n^2\right)^{1/2}=\left(\sum_{i=1}^{n}x_i^2\right)^{1/2} \tag{7-3}$$

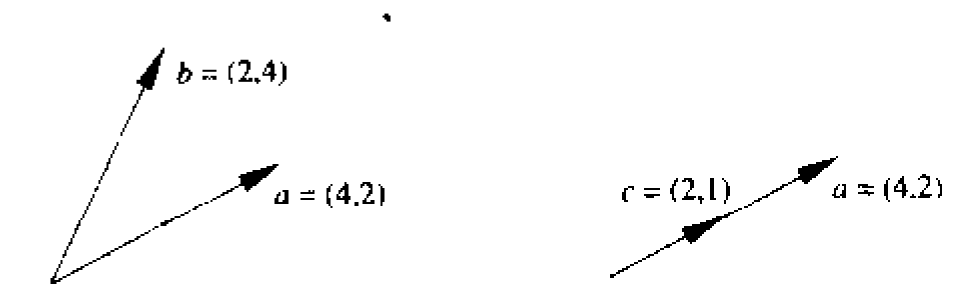


图7-3　比较图中二维向量长度的大小

302

显然，上面的公式是由二维和三维欧几里得几何学推导出来的。$\|\boldsymbol{x}\|_2$中的下标说明范数的类型——这里是L_2，它也可以用内积的形式表示：

$$\|\boldsymbol{x}\|_2=\sqrt{\boldsymbol{x}^T\boldsymbol{x}}$$

有三个向量范数非常有用，它们都是p范数(p norm)家族的成员，p范数的定义如下：

$$\|\boldsymbol{x}\|_p=(|x_1|^p+|x_2|^p+\cdots+|x_n|^p)^{1/p} \tag{7-4}$$

L_2范数是上面p范数的一个例子。向量的L_1范数（p=1）为：

$$\|\boldsymbol{x}\|_1=|x_1|+|x_2|+\cdots+|x_n|=\sum_{i=1}^{n}|x_i| \tag{7-5}$$

向量的L_∞($p=\infty$)范数也称为最大范数如下:

$$\|x\|_\infty = \max(|x_1|,|x_2|,\ldots,|x_n|) = \max_i(|x_i|) \tag{7-6}$$

内置函数norm用于计算向量的p范数。若函数norm只有一个输入变量,那么返回L_2范数:

```
>> u = -2:2;
>> norm(u)
ans =
      3.1623
```

第二个变量用于指定p的值

```
>> norm(u,2)
ans =
      3.1623

>> norm(u,1)
ans =
      6

>> norm(u,Inf)
ans =
      2
```

在最后一个例子中,指定了MATLAB的内部常量Inf,函数返回L_∞范数。 303

例7.2 比较两个向量是否(近似)相等?

在实际中,两个标量α和β在$\alpha \neq 0$且满足下列关系式时,可认为它们相等:

$$\frac{|\alpha-\beta|}{|\alpha|} < \delta$$

这里δ是个很小的公差(参见5.2.3节)。同样地,可用下列式子判断两个向量y和z是否相等:

$$\frac{\|y-z\|}{\|z\|} < \delta \tag{7-7}$$

为了说明上面式子的使用方法,现比较两种求解一个向量的tangent函数值的方法:

```
>> x = linspace(0,2*pi);      %  vector of angles
>> y = sin(x)./cos(x);        %  use array operator to compute tan(x)
>> z = tan(x);
```

用函数plot(x, y, 'o', x, z, '-')可以画出y与x的关系图和z与x的关系图。通过画图可看出这两个向量相等。可以计算出y - z的一个范数进行定量比较。若y和z完全相等,那么‖y - z‖=0。但是,由于计算y和z时会出现舍入误差,要它们相等是不太可能的。实际上,使用方程(7-7)可以测试它们是否相等(注意:测试时可以使用p范数中的任意一个范数):

```
>> norm(y-z,1)/norm(z,1)          %  relative error in the L1 norm
ans =
   7.3821e-17

>> norm(y-z,2)/norm(z,2)          %  relative error in the L2 norm
ans =
   1.0998e-16
```

```
>> norm(y-z,inf)/norm(z,inf)     % relative error in the infinity norm
ans =
   1.1275e-16
```

从上述测试得到下列结论：这两个向量虽不完全相等，但在实际应用中，可以把它们视为相等。

例7.3　由向量构成级数的收敛性

用范数可以判定向量序列的收敛性，这是范数的一个重要应用。例如，在对大型方程组的迭代法求解中，很有必要知道经过迭代后，尝试解是否在不断地接近真值解。这里通过一个例题（本例稍微有点牵强）来说明如何测试向量序列的收敛性。

304

众所周知，几何级数

$$\sum_{k=0}^{\infty}\sigma^k = 1+\sigma+\sigma^2+\cdots$$

在$|\sigma|<1$时，任何情况下都收敛。现在，用几何级数来代替向量的元素，考虑一下会产生什么效果。为明确起见，用$[x]^k$表示向量x的每个元素都进行k次幂运算。则$[x]^k$等价于MATLAB中的数组操作`x.^k`。

有的情况下，向量x中的元素有的比1大，有的比1小。当幂次增加时，向量的大小会不会趋于零？一般情况下，当k增加时，向量x中最大元素的k次幂会大于其他任何元素的k次幂。若最大的元素大于1，那么$[x]^k$也会不断地增加。所以，要使向量级数收敛，那么向量中的所有元素必须小于1。这能够用数值实验证实吗？

程序清单7-1中的函数`vectorSequence`计算了$[x]^k$序列形式的向量的L_1、L_2和L_∞范数，其中$k=1$、2、…、n。函数的可选输入变量是向量x和n，x是序列的起始向量，n是迭代的最大次数。对于向量$x=[1/2\quad 1/4\quad 1/8\quad 1/16]$，它会以很快的速度收敛，输出结果如下：

```
>> vectorSequence
   k    norm(x^k,1)    norm(x^k,2)    norm(x^k,Inf)
   0     4.000e+00      2.000e+00      1.000e+00
   1     9.375e-01      5.762e-01      5.000e-01
   2     3.320e-01      2.582e-01      2.500e-01
  ...      ...            ...            ...
   9     1.957e-03      1.953e-03      1.953e-03
  10     9.775e-04      9.766e-04      9.766e-04
```

最后一列的L_∞范数是几何序列0.5^k，这里0.5是向量x中最大的元素。当k增加时，x^k的L_1和L_2范数的值逐步趋近于其L_∞范数的值。根据上面的推理，我们得出向量x中的最大元素决定了$\|[x]^k\|\to 0$的速度。

若输入向量中有一个或多个元素的值接近1，那么收敛速度将变慢。例如：

```
>> vectorSequence([0.3 0.5 0.7 0.9])
   k    norm(x^k,1)    norm(x^k,2)    norm(x^k,Inf)
   0     4.000e+00      2.000e+00      1.000e+00
   1     2.400e+00      1.281e+00      9.000e-01
   2     1.640e+00      9.833e-01      8.100e-01
  ...      ...            ...            ...
   9     4.297e-01      3.895e-01      3.874e-01
  10     3.779e-01      3.498e-01      3.487e-01
```

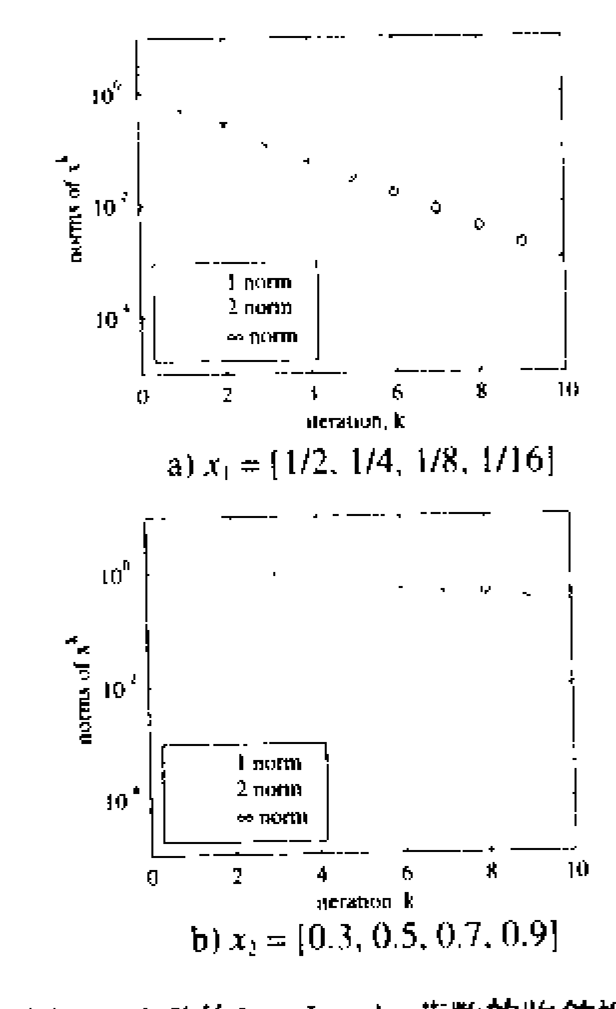
305

程序清单7-1　函数vectorSequence用向量的范数研究序列 $[x]^k$ 的收敛性，其中 x 是一个向量且 $k = 1, 2, \ldots$

```
function vectorSequence(x,n)
% vectorSequence  Behavior of a vector sequence x.^k in different p-norms
%
% Synopsis   vectorSequence
%            vectorSequence(x)
%            vectorSequence(x,n)
%
% Input:     x = (optional) vector used in sequence x.^k
%                Default:  x = [1/2  1/4  1/8  1/16];
%            n = (optional) maximum number of iterations; Default: n = 10
%
% Output:    Plot of norm(x.^k,p) for k = 1,2,...,n and p = 1, 2, Inf

if nargin<1,  x = [1/2  1/4  1/8  1/16];  end
if nargin<2,  n = 10;                     end

fprintf('   k    norm(x^k,1)    norm(x^k,2)    norm(x^k,Inf)\n');
for k=0:n
  y = x.^k;
  norm1 = norm(y,1);  norm2 = norm(y);  normi = norm(y,inf);
  fprintf('%4d  %12.3e   %12.3e   %12.3e\n',k,norm1,norm2,normi);
  semilogy(k,norm1,'d',k,norm2,'o',k,normi,'+');
  hold on
end
hold off
xlabel('iteration, k');  ylabel('norms of x^k');
legend('1 norm','2 norm','\infty norm');
```



a) $x_1 = [1/2, 1/4, 1/8, 1/16]$

b) $x_2 = [0.3, 0.5, 0.7, 0.9]$

图7-4　$[x]^k$ 的 L_1、L_2、L_∞ 范数的收敛性

306
~
307

图7-4分别画出了当x = [1/2 1/4 1/8 1/16] 和x = [0.3 0.5 0.7 0.9]时，序列[x]^{k}的范数。从图中看出，两个向量的三种范数都收敛，但是，向量x= [0.3 0.5 0.7 0.9] 的收敛速度慢一些，而且，L_2的收敛速度小于L_∞大于L_1[⊖]。

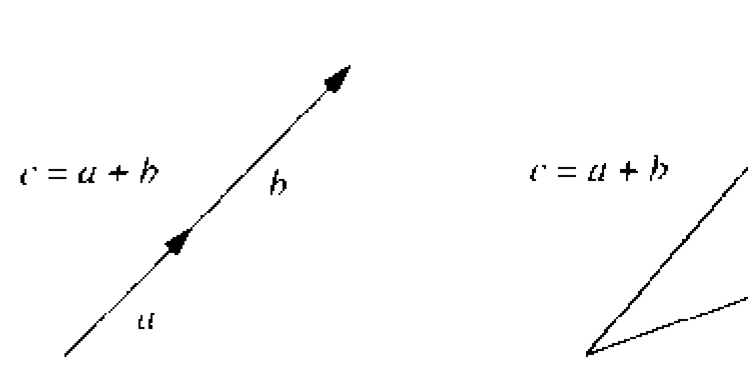
向量范数的数学性质　向量的范数有如下数学性质：

$\|x\|>0$　　对所有$x \neq 0$

$\|\alpha x\| = |\alpha|\,\|x\|$　　对所有α标量

$\|x+y\| \leq \|x\|+\|y\|$　　对任意向量x和y

最后一个性质称为三角不等式，图7-5说明了这一性质。



$\|c\|_2 = \|a\|_2 + \|b\|_2$　　$\|c\|_2 < \|a\|_2 + \|b\|_2$

图7-5　二维向量三角不等式的图形描述

在例7.3中，范数L_1、L_2和L_∞是一致的，它们都可用来测试向量序列[x]^{k}的收敛性。这种性质称为范数等价性（*norm equivalence*），它说明：对那些由实数构成的向量，可以使用它们的p范数中任意一个范数判定收敛性。当用范数计算n维实数向量的大小时，有如下规则（参照文献[32]）：

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty$$

$$\|x\|_\infty \leq \|x\|_1 \leq n\|x\|_\infty$$

这和图7-4的试验结果一致，图7-4说明$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1$。在大多数计算中，我们使用$L_2$范数，当强调计算效率时，可采用$L_\infty$范数。

7.1.3　正交向量

向量内积有它的几何解释。两个非零列向量之间的夹角θ可由下式计算：

$$\cos\theta = \frac{u^T v}{\|u\|_2\|v\|_2} \tag{7-8}$$

计算向量间的任意角度意义不大，计算向量间的特殊夹角，如90度，是比较有意义的。当二维或三维空间中的向量夹角为90度时，称它们垂直或正交（*orthogonal*）。在更高维数空间中的向量垂直还用角度表示会有一些问题，所以采用正交的概念。

两个向量正交意味着什么呢？从方程（7-8）看出，如$\theta=\pi/2$，需有$u^T v = 0$，此时两向量正交。由于方程（7-8）应用于计算任意大小向量间的夹角，因此对于任意大小向量正交的条

⊖　在图7-4中，L_1的范数居上，L_2的范数居中，L_∞的范数居下。

件是当且仅当它们的内积为零。

标准正交向量 正交的单位向量称为标准正交向量。单位向量是指大小为1的向量，而不管它的维数是多少。一个向量除以它的L_2范数就能得到这个向量对应的单位向量，即：

308

$$\hat{\boldsymbol{u}} = \frac{\boldsymbol{u}}{\|\boldsymbol{u}\|}$$

它是在$\boldsymbol{u}$方向上的单位向量。由于

$$\|\boldsymbol{u}\|_2 = \sqrt{\boldsymbol{u}'\boldsymbol{u}}$$

所以，若$\boldsymbol{u}$是单位向量，那么$\boldsymbol{u}'\boldsymbol{u} = 1$。

7.2 矩阵

由实数或复数构成的矩形数组称为矩阵。一个m行n列的矩阵$\boldsymbol{A}$可写成下列形式

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{m1} & & \cdots & a_{mn} \end{bmatrix}$$

矩阵用大写罗马字母表示，矩阵中的元素用小写字母表示，而且每个小写字母带两个下标，第一个下标表示该元素在矩阵中的行号，第二个下标表示该元素在矩阵中的列号。矩阵的维数是由它的总的行数和列数指定的。

在MATLAB中，矩阵和向量惟一的区别就是它们所包含的行数和列数不同。一个标量可视为行数和列数都是1的矩阵。尽管矩阵变量可以取任何有效的MATLAB变量名，但为了保持所用示例的明确性和一致性，我们还是遵守约定，用大写字母表示矩阵，用小写字母表示向量和标量（参见表7-1）。

7.2.1 矩阵中的每行和每列都是向量

矩阵可以看作是行向量或列向量的集合。在行和列的基础上可以有效地实现许多操作。下列语句实现了由三个行向量构成一个矩阵$\boldsymbol{A}$：

```
>> u = 1:4;    v = 5:8;    w = 9:12;
>> A = [u; v; w]
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

这些向量转置后组成另一个由三个列向量组成的矩阵$\boldsymbol{B}$：

309

```
>> B = [u'  v'  w']
B =
     1     5     9
     2     6    10
     3     7    11
     4     8    12
```

需要记住：在MATLAB中，矩阵的列用空格隔开，而矩阵的行用分号隔开。

把矩阵视为向量的集合后，用冒号运算符能够方便地引用矩阵中的行和列（参见2.2.3节）。

例如，用下列代码引用矩阵$\boldsymbol{B}$（上面定义的由三个列向量构成的矩阵$\boldsymbol{B}$）的第一行和第二列：

```
>> B(1,:)
ans =
     1     5     9

>> B(:,2)
ans =
     5
     6
     7
     8
```

用冒号运算符还能够引用矩阵中的子矩阵，子矩阵是指矩阵中的元素不完全包含原矩阵中各行和各列的元素，只是其中的一部分。可把这个子矩阵的值赋给另一个矩阵变量，如下例所示（矩阵$\boldsymbol{B}$如上面的定义）：

```
>> C = B(2:3,1:2)
C =
     2     6
     3     7

>> D = B(3:4,:)
D =
     3     7    11
     4     8    12
```

7.2.2　对矩阵进行的操作

本节介绍矩阵的操作规则，给出了矩阵元素级（element-by-element）的计算公式以及从数学上解释其操作的意义。掌握元素级的计算公式固然重要，但是更应该把重点放在对矩阵行和列的操作上，特别是在矩阵—矩阵和矩阵−向量求积的时候。只有理解了行和列的操作，才能构造出元素级的公式。图7-6总结出对矩阵的高级操作和含元素级的低级操作之间的关系。

310

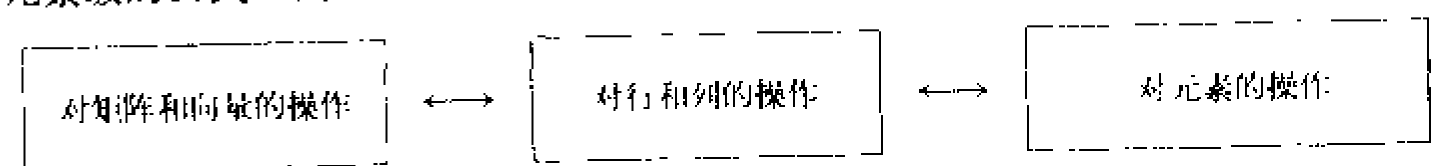


图7-6　说明了矩阵、向量操作的等级。线性代数强大的功能就是能够表示高级概念（左边）并用低级概念（右边）的精确规则来实现这些高级概念，它们之间用行和列操作连接起来

加法与减法　假设$\boldsymbol{A}$、$\boldsymbol{B}$、$\boldsymbol{C}$、$\boldsymbol{D}$都是$m \times n$的矩阵。对它们进行加、减运算实际上是它们对应元素的加、减操作：

$$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B} \Leftrightarrow c_{i,j} = a_{i,j} + b_{i,j} \qquad i = 1,\cdots,m;\quad j = 1,\cdots,n$$

$$\boldsymbol{D} = \boldsymbol{A} - \boldsymbol{B} \Leftrightarrow d_{i,j} = a_{i,j} - b_{i,j} \qquad i = 1,\cdots,m;\quad j = 1,\cdots,n$$

在MATLAB中，对矩阵的加、减就像对标量的加、减，如：

```
>> A = [1 2; 3 4];     B = [5 4; 3 2];
>> C = A+B
C =
     6     6
     6     6
```

```
>> D = A-B
D =
    -4    -2
     0     2
```

与标量相乘 假设A、B都是$m \times n$的矩阵，σ是个标量。计算σA实际上是把矩阵中每个元素乘以σ：

$$B = \sigma A \Leftrightarrow b_{ij} = \sigma a_{ij} \quad i = 1,\cdots,m; \quad j = 1,\cdots,n$$

311

在MATLAB中，标量乘以矩阵很简单：

```
>> A = [1 2; 3 4];    s = 3;
>> s*A
ans =
     3     6
     9    12

>> A*s
ans =
     3     6
     9    12
```

根据上面的计算结果可知：标量和矩阵的乘法满足交换律。

矩阵的转置 向量的转置是指把行向量转置成列向量，或把列向量转置成行向量。对于矩阵的转置，是把矩阵的所有行转置成列，元素级的操作就是把矩阵的行下标和列下标进行交换：

$$B = A' \Leftrightarrow b_{ji} = a_{ij} \quad i = 1,\cdots,m; \quad j = 1,\cdots,n$$

对于方阵，转置就是交换矩阵中关于对角线对称的元素。

在MATLAB中，矩阵的转置符号和向量的转置符号相同，都是撇号。以下是矩阵转置的简单例子：

```
>> A = [0 0 0; 0 0 0; 1 2 3; 0 0 0]
A =
     0     0     0
     0     0     0
     1     2     3
     0     0     0

>> B = A'
B =
     0     0     1     0
     0     0     2     0
     0     0     3     0
```

若$A = A'$，那么A称为对称矩阵（*symmetric matrix*）。

在复数矩阵中（矩阵中的元素为复数值），不使用转置操作，对应于转置的操作称为共轭转置（*conjugate transpose*）、转置伴随矩阵（*adjoint*）或厄密共轭矩阵（*hermitian conjugate*），此时不仅交换关于对角线对称的元素，而且计算每个元素的共轭复数。若A是复数矩阵，共轭转置可表示成A^H[⊖]。对于3×2的矩阵，可写成：

312

⊖ 有些书中用A^*。

$$A=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\\ a_{31} & a_{32}\end{bmatrix} \quad\Rightarrow\quad A^{H}=\begin{bmatrix}\bar{a}_{11} & \bar{a}_{21} & \bar{a}_{31}\\ \bar{a}_{12} & \bar{a}_{22} & \bar{a}_{32}\end{bmatrix}$$

这里 $\bar{a}_{ij}$ 是a_{ij}的共轭复数。在MATLAB中，对矩阵使用撇号运算符也返回矩阵的共轭转置矩阵，如：

```
>> C = sqrt( [1 -4; 9 16; -25 36] )
C =
   1.0000                  0 + 2.0000i
   3.0000             4.0000
        0 + 5.0000i   6.0000

>> D = C'
D =
   1.0000             3.0000                  0 - 5.0000i
        0 - 2.0000i   4.0000             6.0000
```

矩阵-向量的积：**按列乘** 矩阵与向量的积在数值线性代数中非常重要。矩阵A与列向量x的积可以被看作矩阵A中列的线性组合，或看作由矩阵A中行向量与x的一列的内积。我们首先介绍第一个观点：把矩阵与向量的积Ax看作矩阵中的列的线性组合。

线性组合要用到由n列向量组成的向量集$\{a_{(1)}, a_{(2)}, \dots, a_{(n)}\}$。下标用括号括起来说明$a_{(1)}$是集合中的第一个向量，而不是指向量$a$中的第一个元素。对给定的系数集$x_j$，$j=1, \dots, n$；$a_{(j)}$的线性组合构成另一个向量$b$：

$$x_1a_{(1)}+x_2a_{(2)}+\cdots+x_na_{(n)}=b,\ 或\ \sum_{j=1}^{n}a_{(j)}x_j=b \tag{7-9}$$

每个$a_{(j)}$中有m个元素，并且不要求m与n相等。要写出和中的所有项就要使用a的两个下标：

313 a_{ij}表示向量$a_{(j)}$的第i个元素（第i行）。根据这一约定，方程（7-9）可写成：

$$x_1\begin{bmatrix}a_{11}\\ a_{21}\\ \vdots\\ a_{m1}\end{bmatrix}+x_2\begin{bmatrix}a_{12}\\ a_{22}\\ \vdots\\ a_{m2}\end{bmatrix}+\cdots+x_n\begin{bmatrix}a_{1n}\\ a_{2n}\\ \vdots\\ a_{mn}\end{bmatrix}=\begin{bmatrix}b_1\\ b_2\\ \vdots\\ b_m\end{bmatrix}$$

向量b中元素个数m等于每个向量$a_{(j)}$中的元素个数，系数x_j的个数n等于向量$a_{(j)}$的个数。若把$a_{(j)}$写成一个矩阵的形式，那么上面的式子可再简化成下式：

$$\begin{bmatrix} a_{(1)} \Big| a_{(2)} \Big| \cdots \Big| a_{(n)}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}=\begin{bmatrix}b\end{bmatrix}$$

上面的式子说明：系数x_j是一个向量中的元素。详细列出矩阵中的各项与等号右边向量中各项元素后得到：

$$\begin{bmatrix}a_{11} & a_{12} & \cdots & a_{1n}\\ a_{21} & a_{22} & \cdots & a_{2n}\\ \vdots & \vdots & & \vdots\\ a_{m1} & a_{m2} & \cdots & a_{mn}\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ \vdots\\ x_n\end{bmatrix}=\begin{bmatrix}b_1\\ b_2\\ \vdots\\ b_m\end{bmatrix}$$

为了节省空间和避免重复书写下标，上面的方程可简写成

$$Ax = b$$

这里A中的列是线性组合中的向量集，上面的操作说明

> 矩阵-向量的积$b=Ax$是把A中的各列向量线性组合得到一个向量b。

本章后面几节介绍与上面解释等价的对Ax的另一种解释。

为了使计算机能对矩阵的列进行操作，需要引入一个更详细的矩阵-向量乘积公式。把方程(7-9)的行下标清楚地写出来就能达到这一目的。方程（7-9）中的向量是$a_{(j)}$和b，$a_{(j)}$中的第i行的值与b中第i行的值相关，因此 [314]

$$b = Ax \Leftrightarrow b_i = \sum_{j=1}^{n} a_{ij} x_j \quad \text{其中}x\text{和}b\text{都是列向量} \tag{7-10}$$

方程（7-10）说明了矩阵-向量乘积的规则（即$n\times 1$列向量右乘$m\times n$矩阵）。此时，要求矩阵A中的列数等于列向量x的行数。矩阵-向量乘积得到矩阵的维数是：

$$[m\times n]\cdot[n\times 1]=[m\times 1]$$

矩阵A和向量x的内维数（inner dimension）n必须相同，外维数（outer dimension）m和1决定向量b的大小。用算法实现上面的计算过程表示如下：

算法7.1 按列计算的矩阵-向量乘积

initialize: b = zeros(m, 1)
for j = 1, ..., n
  for i = 1, ..., m
    $b(i) = A(i, j)\,x(j) + b(i)$
  end
end

在MATLAB中，用*号运算符能够自动地对匹配的矩阵和向量执行算法7.1。例如：

```
>> A = [1:4;  5:8;  9:12],
A =
     1     2     3     4
     5     6     7     8
     9    10    11    12


>> x = (0.1:0.1:0.4)'
x =
    0.1000
    0.2000
    0.3000
    0.4000
>> b = A*x
b =
    3.0000
    7.0000
   11.0000
```

[315]

使用前面语句定义的矩阵A和向量x，使用冒号运算符也能够进行按列的计算。

```
>> b = x(1)*A(:,1) + x(2)*A(:,2) + x(3)*A(:,3) + x(4)*A(:,4)
b =
    3.0000
    7.0000
   11.0000
```

得到的结果和**A*****x**的结果相同。

7.3.2节介绍了矩阵-向量乘积的另一种解释和使用。例7.5说明矩阵-向量乘积还可当作坐标变换看待。

例7.4 按向量的线性组合计算矩阵-向量乘积

7.1.1节中介绍了使用向量进行线性组合来生成新的向量。例如

$$\boldsymbol{w}=\begin{bmatrix}4\\2\end{bmatrix}=4\begin{bmatrix}1\\0\end{bmatrix}+2\begin{bmatrix}0\\1\end{bmatrix}$$

7.2.2节中把矩阵-向量的乘积解释成矩阵中列向量的线性组合。若把上面例子中等式最右边的两个向量作为一个矩阵中的列向量，那么向量**w**可用下式得到

$$\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}4\\2\end{bmatrix}=\begin{bmatrix}4\\2\end{bmatrix}$$

同样，线性组合

$$\boldsymbol{w}=2\begin{bmatrix}4\\2\end{bmatrix}-4\begin{bmatrix}1\\0\end{bmatrix}-2\begin{bmatrix}0\\1\end{bmatrix}$$

等价于

$$\begin{bmatrix}4&1&0\\2&0&1\end{bmatrix}\begin{bmatrix}2\\-4\\-2\end{bmatrix}=\begin{bmatrix}4\\2\end{bmatrix}$$

316 鼓励读者计算出它们的结果并验证其正确性。

矩阵-向量的乘积：按行乘 把矩阵看作行向量集合的思想与前面把矩阵看作列向量集合的思想是一致的，这两种计算得到的结果完全一样。此时，矩阵-向量的乘积可以看作由矩阵中行向量与被乘向量的内积组成的向量。按列乘的思想有利于加深与矩阵向量操作相关数学知识的理解，而按行乘的思想便于手工计算。

思考一下下列写成矩阵中列的线性组合形式的矩阵-向量乘积：

$$\begin{bmatrix}5&0&0&-1\\-3&4&-7&1\\1&2&3&6\end{bmatrix}\begin{bmatrix}4\\2\\-3\\-1\end{bmatrix}=4\begin{bmatrix}5\\-3\\1\end{bmatrix}+2\begin{bmatrix}0\\4\\2\end{bmatrix}-3\begin{bmatrix}0\\-7\\3\end{bmatrix}-1\begin{bmatrix}-1\\1\\6\end{bmatrix}$$

$$=\begin{bmatrix}(5)(4)&+&(0)(2)&+&(0)(-3)&+&(-1)(-1)\\(-3)(4)&+&(4)(2)&+&(-7)(-3)&+&(1)(-1)\\(1)(4)&+&(2)(2)&+&(3)(-3)&+&(6)(-1)\end{bmatrix}=\begin{bmatrix}21\\16\\-7\end{bmatrix}$$

在第二行中，第一个向量的元素相乘的顺序被颠倒了，上面的计算式说明计算结果所得向量的元素是矩阵的行向量与向量$[4,\ 2,\ -3,\ -1]^T$的内积。如：

$$[5\ \ 0\ \ 0\ \ -1]\begin{bmatrix}4\\2\\-3\\-1\end{bmatrix}=(5)(4)+(0)(2)+(0)(-3)+(-1)(-1)=21$$

根据按行乘的思想，任意3 × 4矩阵A与4 × 1向量x的乘积如下所示

$$\begin{bmatrix}a'_{(1)}\\ \hline a'_{(2)}\\ \hline a'_{(3)}\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}a'_{(1)}\cdot x\\a'_{(2)}\cdot x\\a'_{(3)}\cdot x\end{bmatrix}=\begin{bmatrix}b_1\\b_2\\b_3\end{bmatrix}$$

这里$a'_{(1)}$、$a'_{(2)}$、$a'_{(3)}$是构成矩阵A的行向量；$a'_{(i)}$是指矩阵A的第i行向量，而$a_{(i)}$是指矩阵A的第i列向量，行向量和列向量用“′”符号来区分。总之

317

由A中的行向量与x内积得到的向量就是$b=Ax$的乘积

按行的观点，对于任意大小的矩阵和与其匹配向量的乘积，有如下算法：

算法7.2 矩阵-向量按行乘积

```
initialize: b = zeros(n, 1)
for i = 1, ..., m
   for j = 1, ..., n
     b(i) = A (i, j) x (j)+ b (i)
   end
end
```

算法7.1和算法7.2最本质的区别是它们的循环次序不同。算法7.1外循环的次序是按矩阵A的列来循环，而算法7.2外循环的次序是按矩阵A的行来循环。所以，算法7.2顺序地计算b (i)的值，这在手工计算中比较方便。

对于矩阵A和向量x：

```
>> A = [5 0 0 -1; -3 4 -7 1; 1 2 3 6];    x = [4; 2; -3; -1];
```

按行乘的过程可写成下列形式：

```
>> b = [A(1,:)*x; A(2,:)*x; A(3,:)*x]
b =
    21
    16
    -7
```

这种计算有助于加深理解，但是，计算机程序中却不会用到这种按行乘的方法，因为用“*”运算符在MATLAB中就能够很快（很高效）地计算出矩阵与向量相乘的结果。

矩阵-向量的乘积：按向量乘 按列乘与按行乘的矩阵-向量乘积主要是对行和列的操作。矩阵-向量相乘的另一种解释就是按向量乘，把矩阵看作是作用于向量的转换。把矩阵A乘以列向量x的结果当作是对向量x的拉伸（缩短）或旋转。例如，下列矩阵-向量乘积

318

$$\begin{bmatrix}2 & 0 & 0\\0 & 2 & 0\\0 & 0 & 2\end{bmatrix}\begin{bmatrix}1\\2\\3\end{bmatrix}=\begin{bmatrix}2\\4\\6\end{bmatrix}$$

结果是对列向量的拉伸，然而

$$\begin{bmatrix}0 & -1 & 0\\1 & 0 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}1\\0\\0\end{bmatrix}=\begin{bmatrix}0\\1\\0\end{bmatrix}$$

是对向量的旋转。在许多应用中都会用到旋转和缩放（scaling）操作。

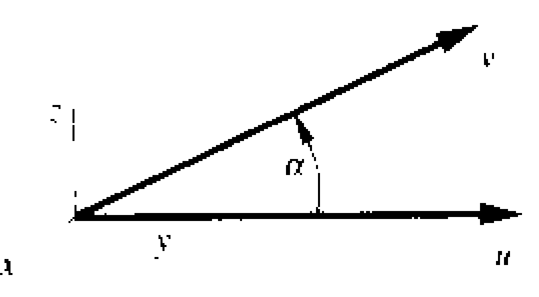
例7.5 遥控臂操作的坐标变换

控制机器人的软件通过给步进电机传送指令，从而移动连杆来控制机器人的动作。步进电机收到控制器指定的角度后就按要求旋转。软件通过计算每个连杆在三个数轴上（在三维空间中）的旋转结果得到连杆最后的位置。用下列三个旋转矩阵$R_x(\alpha)$、$R_y(\beta)$、$R_z(\zeta)$能够完成上面的计算：

$$R_x(\alpha)=\begin{bmatrix}1 & 0 & 0\\0 & \cos\alpha & -\sin\alpha\\0 & \sin\alpha & \cos\alpha\end{bmatrix},\quad R_y(\beta)=\begin{bmatrix}\cos\beta & 0 & \sin\beta\\0 & 1 & 0\\-\sin\beta & 0 & \cos\beta\end{bmatrix}$$

$$R_z(\zeta)=\begin{bmatrix}\cos\zeta & -\sin\zeta & 0\\\sin\zeta & \cos\zeta & 0\\0 & 0 & 1\end{bmatrix}$$

把这三个矩阵都乘以一个包含三个元素的列向量分别使它们绕x、y、z轴旋转α、β和ζ角度，例如，下面的插图描述了向量$\boldsymbol{u}$绕x轴旋转了α角度。向量$\boldsymbol{u}$的起始位置在$y-z$平面上，所以，旋转后的向量$\boldsymbol{v}$仍然在$y-z$平面上。对起始位置不同的向量旋转相同的角度或同一向量旋转不同角度得到的结果都不同。



319

程序清单7-2中的函数rotvec执行了向量旋转的操作。分别用到了三个旋转矩阵：

$$s = R_x\boldsymbol{u},\ t = R_y s,\ \boldsymbol{v} = R_z t$$

它等价于计算出总的变换矩阵$T = R_z R_y R_x$，再乘以向量$\boldsymbol{u}$后得到：

$$\boldsymbol{v} = T\boldsymbol{u} = R_z R_y R_x \boldsymbol{u}$$

程序清单7-2 rotvec函数在三维空间中旋转向量

```
function  v = rotvec(u,alpha,beta,zeta)
% rotvec  Rotates a three dimensional vector
%
% Synopsis:  v = rotvec(u,alpha,beta,zeta)
%
```

```
% Input:      u     = initial vector
%             alpha = angle, in degrees, of rotation about the x-axis
%             beta  = angle, in degrees, of rotation about the y-axis
%             zeta  = angle, in degrees, of rotation about the z-axis
%
%  Output.  v = final vector, i e. result of rotating u through
%               the angles alpha, beta and zeta

a = alpha*pi/180;  b = beta*pi/180;  z = zeta*pi/180;  % convert to radians

% --  set up rotation matrices
Rx = [ 1  0  0; 0  cos(a)  -sin(a); 0  sin(a)  cos(a)];
Ry = [ cos(b)  0  sin(b);  0  1  0; -sin(b)  0  cos(b) ];
Rz = [ cos(z) -sin(z)  0;  sin(z)  cos(z)  0;  0  0  1 ];

v = Rz*Ry*Rx*u;           %  apply the rotation
```

下列语句用来练习使用函数rotvec:

```
>> v = rotvec([1; 0; 0],0,0,90)
v =
     0
     1
     0
>> v = rotvec([1; 0; 0],0,180,0)
v =
    -1
     0
     0

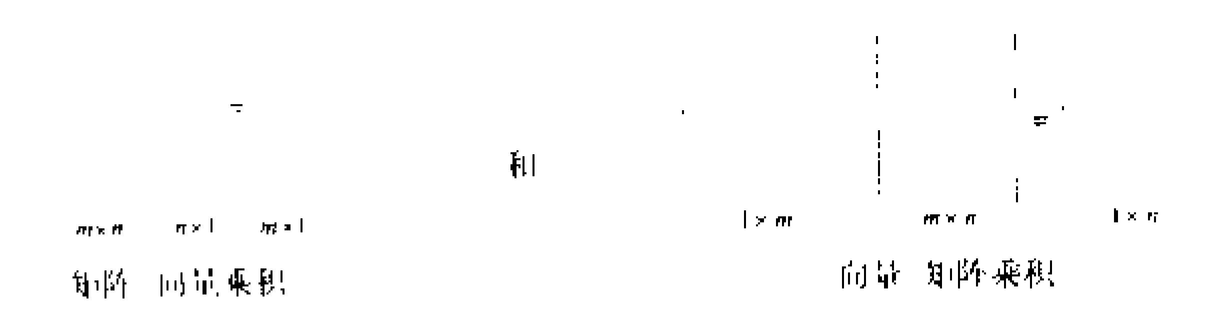
>> v = rotvec([1; 1; 1],45,45,45)
v =
    1.2071
    1.2071
    0.2929
```

320

注意：向量旋转后不改变向量的长度

```
>> u = [-2 3 1]';
>> v = rotvec(u,25,-145,202);
>> norm(v)/norm(u)
ans =
     1
```

向量-矩阵的乘积：按行乘 矩阵与向量乘积中的向量是列向量，矩阵在乘号的左边，而向量在乘号的右边。同样地，向量也能够左乘矩阵，此时，向量是行向量。书中为了区分这两种乘法，分别用“向量-矩阵”和“矩阵-向量”来表示。它们的乘积可表示为



在本书中，要求读者根据向量与矩阵的具体使用来推断是执行何种操作。

对于矩阵-向量的乘积可以看作按行乘或按列乘，向量-矩阵的乘积也是一样，下面将分别讨论这两种思想。为了简化，可把对行与列的操作看作积木式（building-block）的操作。读者需要给每个计算步骤添加详细信息。向量-矩阵乘积的一种解释是：把矩阵中的行向量线性组合得到一个新的向量。假设$\{a'_{(1)}, a'_{(2)}, \ldots, a'_{(m)}\}$是每个向量有$n$个元素的行向量集，它的线性组合为

321

$$u_1 a'_{(1)} + u_2 a'_{(2)} + \cdots + u_m a'_{(m)} = v \tag{7-11}$$

生成包含n个元素的行向量v。若把$a'_{(i)}$构成一个矩阵，系数u_i就是行向量u中的一个元素，那么方程（7-11）中的线性组合等价于向量-矩阵的乘积

$$\begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix} \begin{bmatrix} a'_{(1)} \\ \hline a'_{(2)} \\ \hline \vdots \\ \hline a'_{(m)} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

或

$$u A = v$$

要求u中的列数等于A中的行数。向量v和矩阵A的列数相等。向量-矩阵乘积得到向量的维数是

$$[1 \times m] \cdot [m \times n] = [1 \times n]$$

此时，要求向量与矩阵的内维数相等，此例中都必须是m。

把a_{ij}看作是行向量$a'_{(i)}$的第j个元素，向量-矩阵乘积的形式可写成

$$\begin{bmatrix} u_1 & u_2 & \cdots & u_m \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ a_{m1} & & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$$

向量-矩阵乘积的计算公式是

322

$$uA = v \Leftrightarrow v_j = \sum_{i=1}^{m} u_i a_{ij} \quad u\text{和}v\text{是行向量} \tag{7-12}$$

把上面的公式写成算法，结果如下。

算法7.3　向量-矩阵按行乘

```
initialize: v = zeros (1, n)
for i = 1, ..., m
    for j = 1, ..., n
        v(j) = u(i)A (i, j)+v(j)
    end
end
```

在MATLAB中，用“*”运算符计算向量-矩阵的乘积。例如，向量u和矩阵A的乘积是：

```
>> u = [-3  2  -1]
u =
    -3     2    -1

>> A = [3 2 -4 0; 0 3 -1 2; -3 7 3 -1]
A =
     3     2    -4     0
     0     3    -1     2
    -3     7     3    -1

>> v = u*A
v =
    -6    -7     7     5
```

用算法7.3能够简单明了地说明向量-矩阵乘积的计算过程：

```
>> v = u(1)*A(1,:) + u(2)*A(2,:) + u(3)*A(3,:)
v =
    -6    -7     7     5
```

向量-矩阵乘积：按列乘 对A中各列进行向量-矩阵乘积计算时，得到向量中的每个元素是向量u与矩阵A中对应列的内积。

同上面例子中定义的u和A，把$uA = v$表示成矩阵A中各行的线性组合得到

$$uA = v = -3[3\ \ 2\ \ -4\ \ 0] + 2[0\ \ 3\ \ -1\ \ 2] - 1[-3\ \ 7\ \ 3\ \ -1]$$

由下列计算得到v_1

$$v_1 = (-3)(3) + (2)(0) + (-1)(-3) = u \cdot a_{(1)}$$

323

因此，向量v中的元素是向量u与矩阵A中对应列的内积。任意的1×3向量乘以3×4矩阵得到

$$[u_1\ \ u_2\ \ u_3]\left[a_{(1)} \middle| a_{(2)} \middle| a_{(3)} \middle| a_{(4)}\right] = \left[u \cdot a_{(1)} \middle| u \cdot a_{(2)} \middle| u \cdot a_{(3)} \middle| u \cdot a_{(4)}\right]$$

这里$u \cdot a_{(j)}$是u与A中第j列的内积。在MATLAB中的按列乘算法计算如下所示：

```
>> v = [u*A(:,1)  u*A(:,2)  u*A(:,3)  u*A(:,4)]
v =
    -6    -7     7     5
```

按列计算向量-矩阵乘积的算法如下所示：

算法7.4 按列计算向量-矩阵乘积

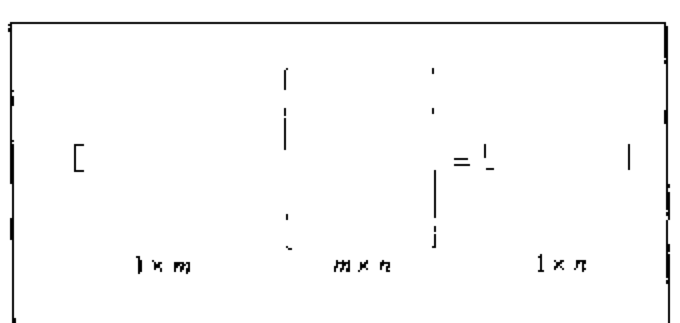
initialize: v = zeros(1, n)
for j = 1, …, n
  for i = 1, …, m
    $v(j) = u(i)A(i, j)+v(j)$
  end
end

对矩阵和向量乘积的总结 前两节详细介绍了两种操作：矩阵-向量乘积和向量-矩阵乘

积。这两种操作不外乎两种情况，按行乘与按列乘，两种情况得到的结果一致[⊖]。一种情况把乘积表示成矩阵中向量的线性组合，便于理解；另一种情况便于手工计算。图7-7总结了这些操作及其组织方法。

$m\times n$ $n\times 1$ $m\times 1$

计算$Ax=v$的两种方法.
1 按A的列计算.
v是A各列的线性组合
2 按A的行计算:
v的每行元素是A的对应行与x的内积



计算$uA=v$的两种方法:
1. 按A的行计算:
v是A各行的线性组合
2 按A的列计算:
v的每列元素是u与A对应列的内积

图7-7 涉及到对矩阵和向量乘积的总结

假设A是一个矩阵，u和x都是与矩阵A匹配的向量，在MATLAB中一般用下列方法计算向量-矩阵乘积和矩阵-向量乘积:

324

```
v = u*A;    % vector-matrix product
y = A*x;    % matrix-vector product
```

MATLAB解释器检测每个操作数的行数和列数并计算出正确的结果。

矩阵-矩阵乘积：按列乘 两个矩阵相乘得到另一个矩阵，它的计算规则是对矩阵-向量乘积和向量-矩阵乘积的按行和按列计算规则的扩展。提出另一种矩阵-矩阵乘积算法其目的不仅仅是只对元素值进行机械地计算。阅读本节内容时，读者要谨记下列3点:

- 按列乘便于数学上的理解
- 按行乘便于手工计算
- 在MATLAB中的“*”运算符处理所有细节

读者在使用“*”运算符的同时不要忽略乘积算法的细节。因为：第一，算法中包含了矩阵匹配性的判断；第二，并非所有的矩阵运算都是数值计算，在矩阵乘法中，有时需要进行符号计算。不论是否拥有能自动进行符号计算的软件包，掌握手工计算方法都是重要的。

325

为了启发矩阵-矩阵乘积的计算，考虑矩阵和两个列向量的乘积。首先介绍按列乘的过程，再介绍按行乘的过程。设计算矩阵-向量乘积Aw和Ax，这里A、w、x分别为

$$A=\begin{bmatrix}4 & 3 & 2\\ -4 & 2 & -2\\ 0 & -1 & 0\\ -1 & 2 & 2\end{bmatrix},\quad w=\begin{bmatrix}1\\ -2\\ 3\end{bmatrix},\ 且\ x=\begin{bmatrix}-3\\ 2\\ 4\end{bmatrix}$$

⊖ 计算的速度取决于循环的顺序，而循环顺序的优劣由矩阵在内存的存储情况，软件优化和计算机内存结构决定。

根据矩阵-向量乘积的规则得到

$$y = Aw = \begin{bmatrix} 4 \\ -14 \\ 2 \\ 1 \end{bmatrix}, \text{ 且 } z = Ax = \begin{bmatrix} 2 \\ 8 \\ -2 \\ 15 \end{bmatrix}$$

然后，把w和x组成一个新的矩阵B:

$$B = [w \ \ x] = \begin{bmatrix} 1 & -3 \\ -2 & 2 \\ 3 & 4 \end{bmatrix}$$

那么，矩阵-矩阵的乘积AB是

$$AB = [y \ \ z] = \begin{bmatrix} 4 & 2 \\ -14 & 8 \\ 2 & -2 \\ 1 & 15 \end{bmatrix}$$

由A和B的积得到一个新的矩阵C，它和矩阵B的列数相同。假设$b_{(j)}$和$c_{(j)}$分别是矩阵B和C的第j列，那么矩阵C中的每一列是由矩阵-向量乘积得到。

$$Ab_{(j)} = c_{(j)} \tag{7-13}$$

如图7-8所示。

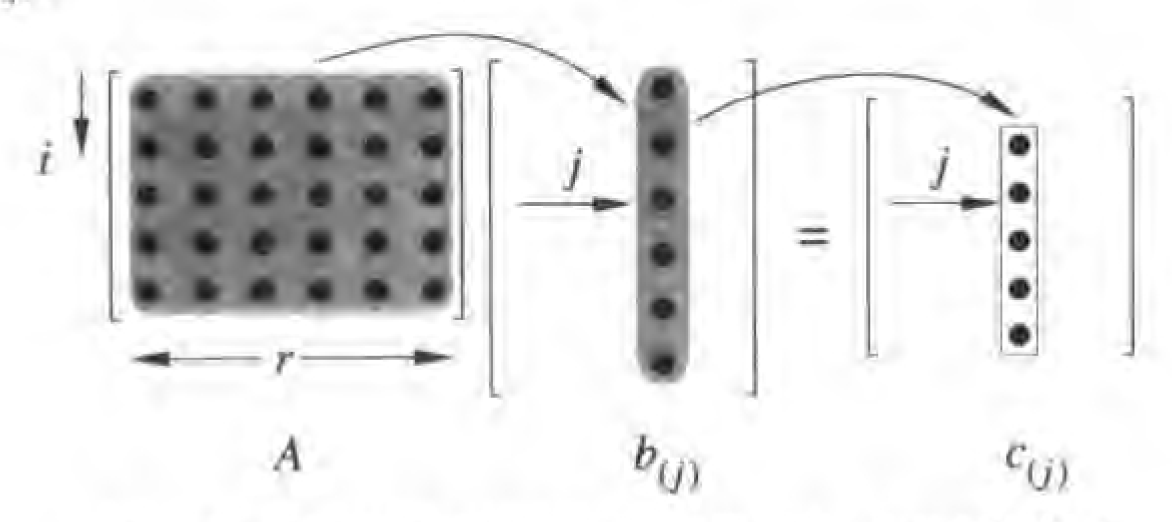


图7-8 矩阵-矩阵乘积的按列乘图示。计算AB时，根据矩阵-向量乘积的规则把A与矩阵B中的每一列相乘

把方程（7-13）中的公式进行扩展并使用双下标得到下式：

$$C = AB \Leftrightarrow c_{(i,j)} = \sum_{k=1}^{r} a_{(i,k)} b_{(k,j)} \tag{7-14}$$

上式中，A是一个$m \times r$矩阵，B是$r \times n$矩阵，结果矩阵C是一个$m \times n$矩阵。 326

矩阵-向量乘积$Ax = y$按列乘的结果y是矩阵A各列的线性组合，矩阵-矩阵乘积按列乘也是这样。$AB = C$得到矩阵C，C中的每一列都是矩阵A中各列的线性组合。图7-8只简要地说明了其结果中的一个线性组合。

矩阵-矩阵乘积的匹配规则与矩阵-向量乘积的匹配规则相同。要使AB有意义，要求矩阵A中的列数等于矩阵B中的行数。两个匹配矩阵进行矩阵-矩阵乘积得到的矩阵维数是

$$[m \times r] \cdot [r \times n] = [m \times n]$$

要实现矩阵-矩阵乘积的计算，就需要详细说明矩阵C中每列元素是如何计算的。可以利用前面提到的矩阵-向量乘积算法中的任意一种（参见算法7.1和7.2）。若用按列乘的算法来求矩阵C中的元素，可得到下列矩阵-矩阵乘积算法，

算法7.5 按列计算矩阵-矩阵乘积

```
initialize: C = zeros(m, n)
for j = 1, ..., n
  for i = 1, ..., m
    for k = 1, ..., r
      C (k, j) = A (k, i)B (i, j)+C (k, j)
    end
  end
end
```

327

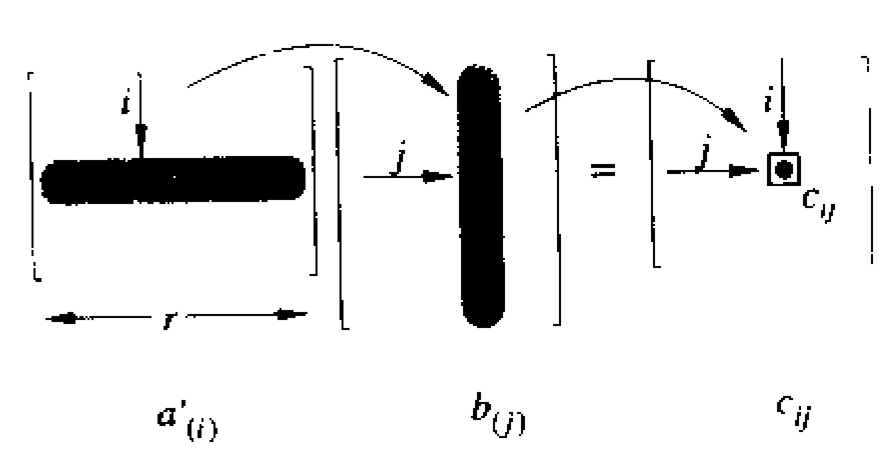
调整循环顺序后（参考文献[32]），有可能可以得到六种求矩阵-矩阵乘积的算法，它们在数学上完全等价。下面将介绍按行乘计算矩阵-矩阵乘积的方法。

矩阵-矩阵乘积：按行乘 图7-8说明矩阵A和矩阵B中第j列向量进行矩阵-向量乘积的结果得到矩阵C的第j列。若采用算法7.2（按行乘的观点计算矩阵-向量乘积的算法）计算$c_{(j)}$中的每个元素，矩阵-矩阵乘积将通过一系列由矩阵A各行与矩阵B各列内积得到，如图7-9所示（只含有产生C中一个元素的图示）。若要计算矩阵C中所有元素值，需要对A的每行和B中每列循环计算，算法如下。

算法7.6 用内积法计算矩阵-矩阵乘积

```
initialize: C = zeros(m, n)
for i = 1, ..., m
  for j = 1, ..., n
    for k = 1, ..., r
      C (i, j) = A (i, k) B (k, j) + C (i, j)
    end
  end
end
```

算法7.6和算法7.5执行完全相同的循环次数，表面上看来，它们只是外循环顺序不同。算法7.6的循环顺序便于手工计算。



328

图7-9 用算法7.6计算$AB = C$的结果图示，其中A是$m \times r$矩阵，B是$r \times n$矩阵

矩阵-矩阵乘积总结　对于两个匹配的矩阵A和B，可以用各种不同的算法求$C = AB$的值。这些算法主要的区别就是对方程（7-14）求和与求积的顺序不同。使用最佳的循环顺序来提高计算效率和许多因素有关，如计算机硬件、用于存储矩阵的基础数据结构和存储矩阵中非零元素的特殊数据结构。MATLAB的设计者为优化计算性能做出了很多努力，所以出现了“*”这一简单的运算符，如：

```
A*B
```

用“*”运算符计算矩阵-矩阵乘积的效率是较高的。

要使AB有意义，要求矩阵A的列数等于矩阵B的行数。两个匹配矩阵进行矩阵-矩阵乘积得到结果矩阵的维数是

$$[m \times r] \cdot [r \times n] = [m \times n]$$

这一规则和矩阵-向量乘积规则相同

$$[m \times n] \cdot [n \times 1] = [m \times 1]$$

向量-矩阵乘积得到结果的维数是

$$[1 \times m] \cdot [m \times n] = [1 \times n]$$

向量内积得到的值为

$$[1 \times n] \cdot [n \times 1] = [1 \times 1]，（一个标量）$$

向量外积得到

$$[m \times 1] \cdot [1 \times n] = [m \times n]$$

所有的匹配规则可总结为：

所有矩阵或向量的乘积都要求操作数的内维数相等。

根据这一规则，MATLAB可以把矩阵、向量和标量都看成同一种数据类型——矩阵。

读者要记住：乘法运算的顺序很重要。乘法运算的顺序不能随便调换。只有当A和B都是方阵而且维数相等时，才能把AB交换顺序；尽管如此，交换后得到的值可能不同了，即，对任意矩阵（进行计算的矩阵是匹配矩阵）　[329]

$$AB \neq BA$$

例7.6　对行和列元素值进行比例缩放

矩阵相乘可以看作是矩阵的转换。例如，$C = DA$可看作是把矩阵A乘一个矩阵D后得到新的矩阵C。从这个例子进行两种变换：缩放矩阵行中的元素值和缩放矩阵列中的元素值。

D为对角矩阵（即$d_{i,j} = 0$，$i \neq j$）时，由于D中不处于对角线上的元素都是零，计算DA就是把D中对角线上的元素值乘以矩阵A每行中的元素。D左乘矩阵A能够缩放矩阵A中行的元素值，右乘可以缩放矩阵A中列的元素值。为了详细地说明，把DA的乘积写成方程（7-14）的形式：

$$C = DA \Leftrightarrow c_{i,j} = \sum_{k=1}^{r} d_{i,k} a_{k,j}$$

求和表达式中只有一项不为零

$$c_{i,j} = d_{i,i} a_{i,j} \quad 因为\ d_{i,k} = 0 \quad i \neq k$$

对所有的j值、$d_{i,i}$的乘数相同，所以，***DA***可以缩放矩阵***A***中行的元素值、右乘***D***可以缩放矩阵***A***中列的元素值：

$$C = AD \Leftrightarrow c_{i,j} = \sum_{k=1}^{r} a_{i,k} d_{k,j} = a_{i,j} d_{j,j}$$

对所有的i值、$d_{j,j}$的乘数相同，所以，***DA***可以缩放矩阵***A***中列的元素值。

用下列MATLAB语句验证上面的结论：

```
>> A = ones(3,3);
>> D = diag([1 2 3]);
>> C1 = D*A
C1 =
     1     1     1
     2     2     2
     3     3     3
>> C2 = A*D
C2 =
     1     2     3
     1     2     3
     1     2     3
```

330

实际中不采用矩阵相乘的方法来缩放矩阵中行、列元素值。当矩阵很大时，计算***DA***的效率非常低。矩阵***D***只有比较少的非零元素，大部分运算都是乘以0及累加和与0值相加，这样很浪费时间。在本例中，***D***是典型的稀疏矩阵（参见附录B）。

表达式***DA***中，***D***是一个对角矩阵，这个表达式是用来说明如何使用线性代数中的概念来简明地描述算法的结果而不用考虑具体实现细节的一个例子。显然，用表达式***C*** = ***DA***比用"把向量***d***中对应元素乘以矩阵***A***中的每一行"进行表示更高效。但是，在表达式具体实现的时候，就要使用后面的表示方法。所以，紧凑的符号表示实际上是算法的缩写，它同样能够表示所要进行的操作而且使用起来更加方便（参见习题20~22）。

7.2.3　矩阵运算和向量运算的操作次数

由于向量与矩阵可以是任意大小，那么有必要注意执行基本运算所需要的计算量。测量计算量最简单的方法是计算算法中浮点操作（floating-point operations）次数（简称为flop）。一次浮点操作就是执行一次加、减、乘或除[⊖]操作。假设每次浮点操作的运行时间相同，那么，浮点操作数大的算法比浮点操作数小的算法所用运行时间长。

含五个元素的列向量***u***和***v***内积结果为：

$$\sigma = \boldsymbol{u}^T \boldsymbol{v} = u_1 v_1 + u_2 v_2 + u_3 v_3 + u_4 v_4 + u_5 v_5$$

表面上看，上面的内积进行了4次加法、5次乘法，也就是9次浮点操作。实际上，用计算机实现上面的内积时，隐藏了一个浮点操作运算，算法如下所示：

331

```
s = 0;
for k=1:length(u)
```

⊖ 有些数值分析人员（如文献[12]）认为一次浮点操作等价于一次乘和加运算。因为线性代数中大部分的操作是$s = s + xy$类型，乘与加同时出现。根据这一规定，内积运算只要执行n次浮点操作，而不是$2n$次浮点操作。约定应用时要注意一致性。本书中，乘与加分开计算。在MATLAB中用函数flops能够求出浮点操作次数。

```
    s = s + u(k)*v(k);
end
```

当u和v都含5个元素时，需要10次浮点操作运算。在循环开始时，u（1）*v（1）与s进行了加运算，所以产生了额外的一次浮点操作。不难看出，对于长度为n的两向量内积，需要$2n$次浮点操作运算。

函数flops会报告执行一系列计算所需要的浮点操作次数[⊖]。可用flops（0）把浮点操作次数重置；否则，函数flops返回从启动MATLAB交互环境开始时系统执行的所有浮点操作次数。下列程序用于求内积的浮点操作次数：

```
>> x = rand(25,1);  y = rand(25,1);
>> flops(0)
>> s = x'*y
s =
    7.4562

>> flops
ans =
   50
```

根据图7-9和方程（7-14）可以明显地看出$C = AB$的元素可以由矩阵A中行与矩阵B中列内积得到。$m \times r$矩阵和$r \times n$矩阵的乘积实际上是进行了$m\ n$次长度为r的内积运算，总的浮点操作次数是$2mnr$次。若两个矩阵都是方阵，那么矩阵-矩阵乘积的浮点操作次数为$2n^3$次，或写为$O(n^3)$。

符号$O(n^3)$读作"n的3次阶"。在讨论矩阵操作的浮点操作次数时，我们只考虑到一位有效数字，不需要更高的精度，因为在实际计算中，运算时间还受诸多因素的影响，如软件和硬件设计等的影响，数量级符号弥补了这一缺点。而且，计算浮点操作次数的目的就是建立操作数大小与计算性能的关系。例如，矩阵-矩阵乘积的浮点操作次数为$O(n^3)$，而向量-向量内积的浮点操作次数为$O(n)$。当n增加时，矩阵-矩阵乘积的运算代价远远大于向量-向量内积的运算代价。表7-2总结了本章提到的矩阵和向量乘积的浮点操作次数。 332

表7-2　向量和矩阵乘积的浮点操作次数

操　作	浮点操作次数	注　释
内积$x'y$	$O(n)$	x和y都是$n \times 1$的向量
外积xy'	$O(n^2)$	x和y都是$n \times 1$的向量
矩阵-向量乘积Ax	$O(n^2)$	A是$n \times n$矩阵，x是$n \times 1$向量
矩阵-矩阵乘积AB	$O(n^3)$	A和B都是$n \times n$矩阵

7.2.4　矩阵的范数

矩阵的范数是矩阵的一种度量，是一个标量，同向量范数一样，这个标量用来度量矩阵

⊖ MATLAB 6不支持flops函数。所以，只能使用LAPACK程序进行矩阵和向量运算。LAPACK程序是在LINPACK和EISPACK的基础上开发的，它是MATLAB的核心。LAPACK基于基础线性代数子程序库（BLAS），计算时更加高效且健壮。由于BLAS子程序不去计算浮点操作次数，所以新版本的MATLAB不会报告执行BLAS的运算代价，对于LAPACK库也是一样的。参照C.Moler，"MATLAB Incorporates LAPACK." MATLAB *News and Notes*，Winter 2000，The Mathworks，Natick，MA。

中元素的大小。一种简单的方法就是应用向量范数来定义矩阵范数。这就要把矩阵改造成向量的形式（例如，把$m \times n$矩阵改成$mn \times 1$向量）来度量矩阵的大小（参照文献[76，第3章]或文献[38，5.6节]）。其中一种采用这种策略的矩阵范数称为Frobenius范数，如下所示：

$$\|A\|_F = \left[\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right]^{1/2} \tag{7-15}$$

为了便于分析矩阵-向量操作，出现了另一种类型的矩阵范数。这种所谓的导出矩阵范数（*induced matrix norm*）用向量范数来定义相关的矩阵范数。

当把矩阵看作线性变换运算符时，我们可以明显地看到它的优点。下列语句

$$y = Ax$$

表明矩阵A把列向量x变换成列向量y。对输入向量x旋转、缩放后，得到输出向量y。现在，从单位向量集中任意选择一个向量x（即所有满足$\|x\|_2 = 1$的向量x），所得向量y完全由矩阵A的内在属性决定。特别地，在所得向量y的L_2范数中，最大的范数表明矩阵A缩放向量的程度。这也是定义矩阵L_2范数的基本思想：

333

$$\|A\|_2 = \max_{\|x\|=1} \|Ax\|_2 \tag{7-16}$$

计算矩阵L_2范数不如计算向量L_2范数简单，而且由于有无限个单位向量，用方程（7-16）直接计算$\|A\|_2$不可行。实际上$\|A\|_2$等价于A^TA最大特征值的平方根（参照文献[32，38]），所以，它常被称为谱范数（spectral norm）。

有两种导出矩阵范数很容易计算，即：1范数和∞范数：

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}| \quad \text{（列求和范数）}$$

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}| \quad \text{（行求和范数）}$$

在MATLAB中，用函数norm计算矩阵的1范数、2范数、∞范数和Frobenius范数，它的句法同向量的范数相同，这里不作深入讨论。

所有的矩阵范数有如下数学性质：

$\|A\| > 0$，对所有$A \neq 0$的矩阵；

$\|\alpha A\| = |\alpha| \, \|A\|$，$\alpha$为任意标量；

$\|A + B\| \le \|A\| + \|B\|$，$A$和$B$为任意两个矩阵。

而且，要求矩阵的范数是相容的：

$\|AB\| \le \|A\| \, \|B\|$，$A$和$B$为任意矩阵，

$\|Ax\| \le \|A\| \, \|x\|$，$A$为任意矩阵，$x$为任意向量。

第二种相容情况是第一种情况的特例，因为可以把列向量看成只有一列的矩阵。

7.3 向量和矩阵的数学性质

本节简要总结矩阵和向量重要的数学性质，将介绍线性无关性，向量空间以及与矩阵行和列相关的向量空间。掌握了本节介绍的内容，能够有助于理解由$Ax = b$定义的方程组（这里A是矩阵，x和b都是列向量）的求解。当读者读到第8章的时候，重新回顾本节内容将有助于学习。

334

为了使读者能够具体地理解所学习内容，例题中的向量只有两到三个元素，以便能够在物理空间中表示这些向量。注意：这里的结论能够扩展到包含任意个元素的向量。

7.3.1 线性无关性

下列两个向量

$$\boldsymbol{u}=\begin{bmatrix}1\\1\\1\end{bmatrix} \quad 和 \boldsymbol{v}=-2\boldsymbol{u}=\begin{bmatrix}-2\\-2\\-2\end{bmatrix}$$

由于上面两个向量处于同一直线上，它们不是线性无关的。每个向量定义了一个方向，在同一方向上，所有向量只是在大小上相差一个倍数。现在，考虑下列两个向量，它们不在同一直线上。

$$\boldsymbol{v}=\begin{bmatrix}-2\\-2\\-2\end{bmatrix} \quad 和 \quad \boldsymbol{w}=\begin{bmatrix}0\\0\\1\end{bmatrix}$$

这两个向量定义了一个平面，这一平面上的任意向量$\boldsymbol{x}$可由这两个向量线性组合得到（即$\boldsymbol{x}=\alpha\boldsymbol{v}+\beta\boldsymbol{w}$，$\alpha$和$\beta$都是标量）。尽管$\boldsymbol{v}$和$\boldsymbol{w}$无关，但$\boldsymbol{x}$与$\boldsymbol{v}$和$\boldsymbol{w}$线性相关（*linearly dependent*）。这种相关性可以推广到任意多个向量。

对于包含n个列向量的向量集$\{\boldsymbol{v}_{(1)}, \boldsymbol{v}_{(2)}, \ldots, \boldsymbol{v}_{(n)}\}$，每个向量都包含$m$个元素。注意$\boldsymbol{v}_{(i)}$用来表示向量集中第$i$个向量，而不是某个向量$\boldsymbol{v}$的第$i$个元素。当集合中的任意向量$\boldsymbol{v}_{(i)}$不能由剩余向量线性组合得到时，我们称整个集合是线性无关的。更准确地说，若当且仅当所有标量系数α_i等于零，下列式子才会等于零，称向量集是线性无关的：

$$\alpha_1\boldsymbol{v}_{(1)}+\alpha_2\boldsymbol{v}_{(2)}+\cdots+\alpha_n\boldsymbol{v}_{(n)}=0 \tag{7-17}$$

否则，若方程（7-17）中存在至少有一个标量系数不为零，那么称向量集是线性相关的。

线性无关性在正交向量中容易看到，例如下列向量

$$\begin{bmatrix}4\\0\\0\\0\end{bmatrix}, \quad \begin{bmatrix}0\\-3\\0\\0\end{bmatrix}, 和 \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

335

都是线性无关的。但是，不是所有线性无关的向量都是正交向量。如，下列向量集线性无关：

$$\begin{bmatrix}1\\-2\\1\end{bmatrix}, \quad \begin{bmatrix}1\\0\\4\end{bmatrix}, 和 \begin{bmatrix}2\\-1\\0\end{bmatrix} \tag{7-18}$$

方程（7-17）定义了向量集线性无关的准则。若取出矩阵的每列构成一个向量集$\{\boldsymbol{v}_{(1)}, \boldsymbol{v}_{(2)}, \ldots, \boldsymbol{v}_{(n)}\}$，那么方程（7-17）等价于下列矩阵-向量乘积：

$$\begin{bmatrix} \boldsymbol{v}_{(1)} & \boldsymbol{v}_{(2)} & \cdots & \boldsymbol{v}_{(n)} \end{bmatrix}\begin{bmatrix}\alpha_1\\\alpha_2\\\vdots\\\alpha_n\end{bmatrix}=\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix} \tag{7-19}$$

上式说明了矩阵中各列线性无关的条件。

> **$m \times n$矩阵A中各列线性无关的条件是：当且仅当$x=(0,0,\ldots,0)^T$是惟一满足$Ax=0$的n维列向量**

要判断矩阵A中的各列是否线性无关，需要用第8章介绍的方法求解$Ax = 0$方程组。若求不出解，说明A中各列不是线性无关的。用7.3.4节介绍的函数`rank`也能够求解方程组$Ax = 0$。

7.3.2 向量空间

单独地考虑一个向量并不常见。根据向量包含元素的个数可把向量分组。例如，对于三维空间中的描述，我们使用包含3个元素的向量，其中的每个元素对应一个(x, y, z)坐标方向。包含3个元素的向量组不同于包含2，4，15或3000个元素的向量组。要能够区分不同的向量组中的向量。

向量空间（*vector space*）是指在一个向量组中，所有的向量中元素个数相等，而且遵循为该空间定义的加法与乘法规则（参照文献[71，72]）。特别地，在同一向量空间中，任一向量对所有向量线性组合得到的结果都是封闭的（还是本向量空间中的向量）。任意两个标量α和β、向量u和v处于同一向量空间，那么线性组合$w = \alpha u+\beta v$也在这一向量空间中。换句话说，一个封闭的向量空间包含所有可能的线性组合。

336

三个元素都是实数的向量构成的集合就是向量空间$\mathbf{R}^3$。"3"表示元素的个数（组件），$\mathbf{R}$表示所有元素都是实数。$\mathbf{C}^3$表示向量中的元素都是复数，而且每个向量包含3个元素。

向量空间$\mathbf{R}^1$、$\mathbf{R}^2$和$\mathbf{R}^3$能直接用几何方法解释。向量空间$\mathbf{R}^1$中的所有元素都在一条线上（实数轴）。$\mathbf{R}^2$空间的元素在一个平面上，如图7-2中的向量处在$\mathbf{R}^2$空间中。向量空间$\mathbf{R}^n$中的所有向量含n个元素，而且每个元素都是实数。根据约定，$\mathbf{R}^n$空间是包含n个元素的列向量。

子空间 子空间（*subspace*）是指包含在另一个向量空间中的向量空间。通常，$\mathbf{R}^n$的子空间是$\mathbf{R}^n$的子集，也是一个向量空间。所以，子空间对其中的所有向量也要求是封闭的。

对于$\mathbf{R}^3$中的向量，假设每个向量的尾部都在坐标原点$(0，0，0)^T$处，那么，向量中的元素可以解释为对应$(x，y，z)$坐标是它们的顶部。图7-10画出了下列3个向量的图示：

$$u=\begin{bmatrix}1\\2\\0\end{bmatrix},\quad v=\begin{bmatrix}-2\\1\\3\end{bmatrix},\text{和 } w=\begin{bmatrix}3\\1\\-3\end{bmatrix}$$

尽管u、v、w包含3个元素，但是它们都处于同一平面中，也就是说它们处在$\mathbf{R}^3$的子空间中。

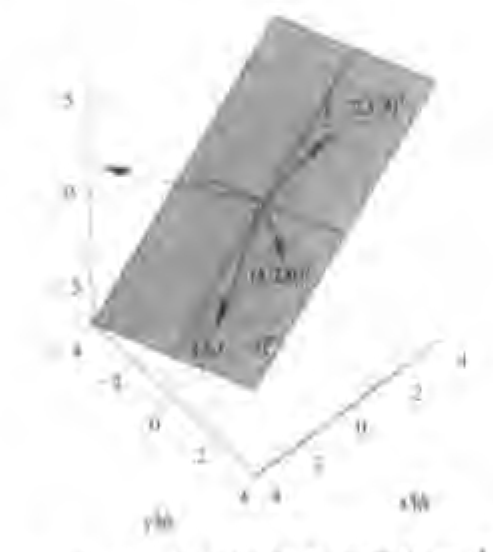


图7-10 $\mathbf{R}^3$的子空间中包含3个元素的向量，未标注的向量是平面的法线

生成子空间（span of subspace） 向量集$\{v_{(1)}, v_{(2)}, \ldots, v_{(n)}\}$通过线性组合可以用来构造任意向量。若向量$w$由下列线性组合得到：

$$\beta_1 v_{(1)}+\beta_2 v_{(2)}+\cdots+\beta_n v_{(n)}=w$$

其中β_i都是标量，就可以说构成的向量w是在由向量集$\{v_{(1)}, v_{(2)}, \ldots, v_{(n)}\}$生成的子空间中。若$v_{(i)}$是$\mathbf{R}^m$空间的向量，那么$v_{(i)}$生成的子空间也是$\mathbf{R}^m$的子空间。若$n \geqslant m$，那么$v_{(i)}$可能生成$\mathbf{R}^m$，但是不能保证一

定会生成。

生成子空间的向量不必是线性无关的，只要求向量线性组合后能够得到子空间中的所有向量。

子空间的基和维数 子空间的基（*basis*）是指一组线性无关的向量，它们用于生成子空间。由于基集是线性无关的，所以要求基集包含最少的向量数。基集中的向量个数称为子空间的维数（*dimension*）。$\mathbf{R}^n$空间的基中有n个线性无关的向量。任意包含n个元素的n个线性无关的向量集构成$\mathbf{R}^n$的基。

337

基和维的概念可用于子空间和整个的空间中，如$\mathbf{R}^n$。因此，图7-10中的3个向量构成了$\mathbf{R}^4$空间中的三维子空间，而且这3个向量构成了这个子空间的基。尽管它们生成了一个三维子空间，但是，从它们包含四个元素可以明显看出它们不是$\mathbf{R}^3$空间中的向量。

正交向量，如：

$$\boldsymbol{a}=\begin{bmatrix}1\\0\\0\end{bmatrix},\qquad \boldsymbol{b}=\begin{bmatrix}0\\1\\0\end{bmatrix},\ 和\ \boldsymbol{c}=\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{7-20}$$

就是最简单的基向量，但是，不是所有的基向量都是正交向量，如，下列集合是$\mathbf{R}^4$的基：

$$\boldsymbol{r}=\begin{bmatrix}4\\0\\2\\-3\end{bmatrix},\quad \boldsymbol{s}=\begin{bmatrix}1\\3\\1\\1\end{bmatrix},\quad \boldsymbol{t}=\begin{bmatrix}2\\1\\7\\2\end{bmatrix},\quad 和\quad \boldsymbol{u}=\begin{bmatrix}4\\-2\\-1\\5\end{bmatrix} \tag{7-21}$$

338

7.3.3 与矩阵相关的子空间

任意的矩阵都有四个子空间，它们分别是列空间、行空间、零空间（null space）和左零空间。这里简要介绍列空间和零空间（参照文献[71、72]可以获得更多的信息）。

列空间 矩阵-向量乘积$\boldsymbol{y}=\boldsymbol{Ax}$可以解释为把矩阵$\boldsymbol{A}$的列线性组合后得到向量$\boldsymbol{y}$。所以，根据子空间基的概念得到，矩阵$\boldsymbol{A}$的列称为矩阵$\boldsymbol{A}$的列空间(column space)或矩阵$\boldsymbol{A}$的范围(range)。不同的线性组合得到不同的向量$\boldsymbol{y}$（即改变向量$\boldsymbol{x}$的元素）。另一方面，若$\boldsymbol{y}$不在矩阵$\boldsymbol{A}$的列空间中，那么不存在$\boldsymbol{x}$，使$\boldsymbol{Ax}$的积为$\boldsymbol{y}$。

若$\boldsymbol{A}$是$m\times n$的矩阵，那么它的列空间的维数小于或等于m。由于矩阵$\boldsymbol{A}$中列的长度是m，所以$\boldsymbol{A}$中各列向量在$\mathbf{R}^m$的子空间中。列空间的维数等于$\boldsymbol{A}$中线性无关列的个数。若矩阵$\boldsymbol{A}$中有n列，那么列空间的最大维数不会超过n。若$n>m$，那么矩阵$\boldsymbol{A}$中的列向量线性相关。

矩阵

$$\boldsymbol{A}=\begin{bmatrix}1&2&0\\-3&-2&-4\\-2&0&2\end{bmatrix} \tag{7-22}$$

由线性无关的三个列向量组成。因此，矩阵$\boldsymbol{A}$的列空间的维是3，且矩阵$\boldsymbol{A}$的列在$\mathbf{R}^3$范围内。

矩阵

$$\boldsymbol{B}=\begin{bmatrix}1&-2&3\\2&1&1\\0&3&-3\end{bmatrix} \tag{7-23}$$

只有两个线性无关的列（第三列可用第一列减去第二列得到），所以，B的列空间的维是2。如果在三维物理空间中表示这三个列向量，它们都在同一平面上，如图7-10所示。

零空间　矩阵A的零空间记做null(A)，它是所有满足$Ax = 0$的向量集。至少存在一个向量满足方程，即：$x = 0$。若A为$m \times n$矩阵，那么A的零空间是$\mathbf{R}^n$的子空间（不是$\mathbf{R}^m$）。要使

339

$y=Ax$中所有元素为零，算法7.2（按行乘的矩阵-向量乘积）说明满足$Ax = 0$的x向量必须与矩阵A的行向量正交。

例如，求方程（7-22）和（7-23）定义的矩阵的零空间。方程（7-22）定义的矩阵A有三个线性无关的列向量，根据方程（7-19），满足$Ax = 0$的解只有一个，即$x = 0$，所以，矩阵A的零空间只有一个零向量。方程（7-23）定义的矩阵B只有两个线性无关的列向量，$Bx = 0$的一个解是$x = [1,\ \ 1,\ -1]^T$，直接替换就能够验证这一计算结果的正确性。这一向量的倍数也是方程$Bx = 0$的解，所以，向量B的零空间是通过$[0,\ 0,\ 0]^T$和$[1,\ \ 1,\ -1]^T$的直线。它是B的行向量所定义平面的法线（参照例7.7）。

内置函数`null`用于求矩阵的零空间。它有两种形式

```
Z = null(A)
Z = null(A,'r')
```

对于给定的矩阵A，函数`null(A)`返回A的零空间的正交基给矩阵Z中各列。用奇异值分解（*Singular Value Decomposition*）算法可以得到A的正交基。函数`null`的另一种形式`Z=null(A, 'r')`生成A的零空间并把结果返回给矩阵Z的各列，而且Z中各元素的值是整数的比值（如有理数）。文献[72,3.2节]介绍了函数`null(A, 'r')`使用的算法。`null(A)`形式得到的结果比较可靠；而`null(A, 'r')`形式对于小型矩阵A，输出的可视化效果好。

例7.7　`null(A)`与A的行空间正交

与列向量生成列空间相同，行向量生成行空间（*row space*）。若矩阵A的零空间中的向量满足$Ax = 0$，那么这些向量与A的行空间正交。这就是图7-10中所画平面的法线的产生方法。

假设B是方程（7-23）定义的矩阵：

```
>> B = [1  -2  3;  2  1  1;  0  3  -3];
```

图7-10说明矩阵B的各个列向量在一个平面中。向量

```
>> Z = null(B)
Z =
    0.5774
   -0.5774
   -0.5774
```

340

在B的零空间中，但与B的列向量并不正交。例如：

```
>> B(:,1)'*Z
ans =
   -0.5774
```

要找到与B的列空间正交的子空间，必需计算B^T的零空间。`null(B')`中的向量与B^T中的行向量（它们实际上是B的列向量）正交，例如：

```
>> Zr = null(B','r')
Zr =
    1.2000
   -0.6000
    1.0000
```

它是图7-10中平面的法线。

7.3.4 矩阵的秩

矩阵的秩（*rank*）是指矩阵中线性无关的列向量的个数，它等于列空间的维数，也等于矩阵中线性无关行向量的个数，因此也等于行空间的维数。

$m \times n$矩阵的秩是r，通常$r \leqslant \min\{m, n\}$。若$r = \min\{m, n\}$，称矩阵为满秩（*full rank*）。若$r < \min\{m, n\}$，称矩阵为秩亏（*rank deficient*）。简单矩阵的秩通过观察可以得到。这里是一些很明显的秩亏[⊖]矩阵的例子：

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 1 & 1 \end{bmatrix}$$

大体上，通过基本的行操作（高斯消去）把矩阵简化为一个上三角矩阵U从而能够得到矩阵的秩。Strang在文献[72，3.2节]详细的描述了这一过程。得到上三角矩阵U后，秩就是在对角线上的值为非零的列向量的个数。实际上，由于舍入误差的影响会导致求秩的计算中丢失精度，对于大型矩阵情况会变得更严重（参照文献[28，5.8节]能获得更详细更全面的讨论信息）。 341

内置函数rank用奇异值分解算法估算矩阵的秩：

```
>> rank([1 2; 1 2])
ans =
     1

>> rank([1 0 1; 2 1 0; 3 1 1])
ans =
     2
```

Gill等人在文献[28，5.8节]中，Golub和Van Loan在文献[32，5.5.8节]中都介绍了计算时的算法。感兴趣的读者可以通过阅读函数rank的代码了解求秩在MATLAB中的计算过程。

7.3.5 矩阵的行列式

对于每个方阵（即$n \times n$方阵），存在惟一的标量值与之对应，该标量值称为矩阵的行列式（*determinant*），记为det(A)。当矩阵A写成矩形的形式时，用竖线代替圆括号或方括号的形式来表示行列式，例如，矩阵A是2×2的方阵，那么

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21} \tag{7-24}$$

对于更大型的矩阵，通过转换成更小的行列式，直到得到能够按上式直接计算的2×2矩阵的形式为止，最后得到整个行列式的值。下面会介绍这一过程。

行列式在理论分析中有重要的性质。但是，在数值计算中，它没有什么价值。计算一个中等规模大小的矩阵的行列式值其计算代价很大，还可能会导致上溢。

在手工计算中，矩阵的行列式可以写成矩阵元素与其对应代数余子式（*minor*）乘积的递归和的形式，代数余子式是指与原矩阵的某一元素对应的子矩阵所形成的行列式。子矩阵是

⊖ 提示：对于矩阵B，把第三列与第二列加两次，或者把第一行与第二行相加。

把原矩阵中要讨论的元素所在的行和列单独拿出来后所剩元素构成的新矩阵。用按行展开余子式计算矩阵的行列式的一般规则如下所示：

$$\det(\boldsymbol{A})=\sum_{j=1}^{n}a_{ij}(-1)^{i+j}M_{ij}\text{，对任意的}i \tag{7-25}$$

这里的余子式M_{ij}，是把矩阵$\boldsymbol{A}$的第i行、第j列去掉后得到的$(n-1)\times(n-1)$行列式。上面计算$\det(\boldsymbol{A})$是对一行上所有元素与对应的列子式乘积的和。用按列展开余子式计算矩阵的行列式的一般规则如下所示：

342

$$\det(\boldsymbol{A})=\sum_{i=1}^{n}a_{ij}(-1)^{i+j}M_{ij}\text{，对任意的}j \tag{7-26}$$

图7-11中列出了与3 × 3矩阵第一行元素对应的3个余子式。求此3 × 3矩阵的行列式时，只要在第一行展开各个余子式：

$$\begin{aligned}\begin{vmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{vmatrix} &= a_{11}(-1)^{1+1}\begin{vmatrix}a_{22} & a_{23}\\ a_{32} & a_{33}\end{vmatrix}+a_{12}(-1)^{1+2}\begin{vmatrix}a_{21} & a_{23}\\ a_{31} & a_{33}\end{vmatrix}\\ &\quad +a_{13}(-1)^{1+3}\begin{vmatrix}a_{21} & a_{22}\\ a_{31} & a_{32}\end{vmatrix}\\ &= a_{11}M_{11}-a_{12}M_{12}+a_{13}M_{13}\end{aligned} \tag{7-27}$$

利用方程（7-24）计算M_{11}、M_{12}、M_{13}并求出上面表达式的值。对于大型的矩阵，用递归法展开余子式（方程（7-25）和方程（7-26）），直到最后得到2 × 2矩阵的行列式的形式。

$$\begin{pmatrix}\boxed{a_{11}} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}\quad \begin{pmatrix}a_{11} & \boxed{a_{12}} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}\quad \begin{pmatrix}a_{11} & a_{12} & \boxed{a_{13}}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{pmatrix}$$

$$M_{11}=\begin{vmatrix}a_{22} & a_{23}\\ a_{32} & a_{33}\end{vmatrix}\quad M_{12}=\begin{vmatrix}a_{21} & a_{23}\\ a_{31} & a_{33}\end{vmatrix}\quad M_{13}=\begin{vmatrix}a_{21} & a_{22}\\ a_{31} & a_{32}\end{vmatrix}$$

图7-11　3 × 3矩阵的第一行的余子式，对应图中阴影部分

按余子式展开的方法只针对小型矩阵，当矩阵变大时，计算的代价也增加得很快。假设$fp(n)$是按余子式展开矩阵求对应的行列式所需的浮点操作次数。根据方程（7-24）、（7-27）和（7-25）得到的浮点操作次数为

343

$$\begin{aligned}&fp(2)=3,\\ &fp(3)=3\,[fp(2)+2]=15,\\ &fp(4)=4\,[fp(3)+2]=4\{3\,[fp(2)+2]+2\},\\ &\vdots\\ &fp(n)=n\,[fp(n-1)+2]\sim O(n!).\end{aligned}$$

当n很大时，需要的计算量将相当大。

线性方程组$\boldsymbol{Ax}=\boldsymbol{b}$的形式解[⊖]可以用克拉默法则（Cramer's Rule）来表示：解向量的第i个

⊖　第8章介绍线性方程组的求解。

元素是

$$x_i = \frac{\det(A_i)}{\det(A)} \tag{7-28}$$

其中A_i是用等号右边的向量b替换矩阵A中对应的第i列后所得到的矩阵。用克拉默法则求解x时需要计算$(n+1)$次$n \times n$矩阵的行列式。当n为中等大小的数时（如$n > 5$），这种方法效率很低。方程（7-28）的简洁表达形式掩盖了实际求解的低效性。

若用克拉默法则和其他det(A)计算方法不能求解方程组时，可以看出，det(A)的值至少具有诊断价值。根据方程（7-28），当det(A) = 0时，不能用克拉默法则求解。当满足det(A) ≠ 0时，方程组才有解。由于求det(A)耗费很多时间(占求解方程组一半的时间)，最好在求解$Ax=b$之前检查det(A)的数量级。

有些含很多零的矩阵求行列式特别方便。对于4 × 4对角矩阵，利用方程（7-25）得到

$$\begin{vmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & d_3 & 0 \\ 0 & 0 & 0 & d_4 \end{vmatrix} = d_1 d_2 d_3 d_4 \tag{7-29}$$

求三角矩阵对应的行列式也很方便。上三角矩阵只有主对角线上方的元素非零（$j \geq i$，j是列下标，i是行下标），下三角矩阵只有主对角线下方的元素非零（$i \geq j$）。例如，4 × 4的上三角矩阵对应的行列式

344

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{vmatrix} = a_{11} a_{22} a_{33} a_{44} \tag{7-30}$$

通常，对于$n \times n$上三角矩阵或下三角矩阵，它们对应的行列式都是

$$\det(A) = a_{11} a_{22} \ldots a_{nn} \qquad (A\text{只可为三角矩阵})$$

把方阵分解成两个三角矩阵相乘能够提高求解行列式的效率。这就是所谓的LU分解。把矩阵A分解成下三角矩阵L和上三角矩阵U得到[⊖]

$$A = LU$$

任意给定两个方阵B和C，BC的行列式等于方阵B的行列式与方阵C的行列式的积，即det(BC)=det(B)det(C)(参照文献[12，78])。因此，若$A = LU$，那么

$$\det(A) = \det(L)\det(U) = (\ell_{11}\ell_{22}\ldots\ell_{nn})(u_{11}u_{22}\ldots u_{nn})$$

分解矩阵A成LU的形式需要$O(n^3)$次浮点操作，然后计算det(L)和det(U)需要$2n$次浮点操作。相比之下，$2n$次的浮点操作可以忽略不计。因此，计算A的行列式需要$O(n^3)$次浮点操作。这种分解方法比展开余子式的方法要高效得多（最小的矩阵除外）。MATLAB内置函数det用分解成LU形式的方法求方阵的行列式。

虽然分解方法很高效，但在实际的数值算法中仍很少用到行列式。求解线性方程组时，人们更喜欢使用LU分解法计算行列式。若方程组是奇异的（即若det(A) = 0），那么计算L和U

⊖ 注意L和U的计算并非是简单地提取出A的上三角和下三角部分。

时将会出现除数为零的情况从而导致计算失败。这就要求在计算det(*A*)之前就要让计算机知道det(*A*) = 0。

当det(*A*)的值很大时，计算中还会出现上溢，这也是一个隐含的问题。假设一个128 × 128的三角矩阵，对角线上的元素都为1000，那么最后的结果是$1000^{128}=10^{384}$，它比计算机能表示的最大浮点数`realmax`都大，目前，计算机还无法表示它。要知道这还是一个容易计算的矩阵。

345

行列式也说明，用数学符号表示线性代数的思想和用计算机的程序实现的步骤是不同的（想知道更多关于行列式的局限性的读者可以参照文献[28，3.6节]和文献[68，第10章]）。

7.4 特殊矩阵

有些类型的矩阵在线性代数的计算中有特殊的应用。它们的计算规则同一般的矩阵相同，而且这些矩阵能够简化计算。本节介绍常见的"特殊"矩阵和MATLAB中创建这些矩阵的命令。

7.4.1 对角矩阵

假设i是矩阵元素的行下标，j是矩阵元素的列下标，所有$i \neq j$对应元素都为零的矩阵是对角矩阵。所以，$n \times n$对角矩阵只会有n个非零元素。此时，用一个包含对角线元素的n维向量能够很方便地表示该对角矩阵。数学符号是

$$C = \mathrm{diag}(c_1, c_2, \cdots, c_n) = \begin{bmatrix} c_1 & 0 & \cdots & 0 \\ 0 & c_2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & c_n \end{bmatrix}$$

MATLAB函数`diag`可由向量创建一个对角矩阵或从一个已知矩阵中提取出对角元素生成一个变量。若函数`diag`的输入变量是一个向量，MATLAB利用这个向量创建对角矩阵：

```
>> x = [1 -5  2  6];
>> A = diag(x)
A =
     1     0     0     0
     0    -5     0     0
     0     0     2     0
     0     0     0     6
```

346

若函数`diag`的输入变量是一个矩阵，那么结果得到由对角线元素构成的向量：

```
>> B = [1 -5  2; 6  12  -4; 0  7  9];
>> y = diag(B);
y =
     1
    12
     9
```

用对角矩阵乘以矩阵*A*能够缩放矩阵*A*中行或列中的元素值（参照例7.6）。

7.4.2 单位矩阵

在标量计算中，$\alpha \times 1$的结果还是α；同样，在线性代数中，*AI*的结果还是*A*，*I*称为单位矩阵（*identity matrix*）。单位矩阵是主对角线上的元素都为1的对角矩阵。

单位矩阵是方阵，它的行数等于列数。要使AI有效，要求A和I都满足矩阵相乘的条件，即，I的行数等于A的列数。因此，单位矩阵在计算中的大小根据上下文确定。有些情况下，通过给矩阵I添加下标来表示矩阵的行数（和列数）。例如，4 × 4单位矩阵可以写为：

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

这种使用单下标来表示单位矩阵大小的方法是矩阵表示法的一个特例。

函数eye可用于创建单位矩阵，它可以带一个或两个输入变量。

```
I = eye(n)
I = eye(m,n)
```

带一个输入变量的形式中，输入变量表示单位方阵的行数和列数。带两个输入变量的形式中，m表示矩阵的行数，n表示矩阵的列数，这种类似单位方阵的矩形矩阵在主对角线上的元素都是1。 347

7.4.3 矩阵的逆

假设A是方阵（即$n \times n$），它的元素都是实数。那么A的逆（*inverse*）写成A^{-1}，它有如下性质：

$$A^{-1}A = I \qquad 和 \qquad AA^{-1} = I$$

矩阵A的逆可能存在，也可能不存在。当A^{-1}存在时，称A可逆（*invertible*）或非奇异的（*nonsingular*）；否则，A是奇异（*singular*）的。

求矩阵的逆需要求解线性方程组，这将在第8章详细介绍。现在，来考虑下列方程

$$Ax = b$$

A是$n \times n$方阵，x和b都是包含n个元素的列向量。假设A和b已知，x未知，在方程左右两边都左乘A^{-1}得

$$A^{-1}Ax = A^{-1}b \rightarrow Ix = A^{-1}b \rightarrow x = A^{-1}b$$

得到方程$Ax = b$的解是$x = A^{-1}b$。

内置函数inv用于计算一个矩阵的逆：

```
>> A = [2 -1 0 0; -1 2 -1 0, 0 -1 2 -1; 0 0 -1 2]
>> Ainv = inv(A)
Ainv =
    0.8000    0.6000    0.4000    0.2000
    0.6000    1.2000    0.8000    0.4000
    0.4000    0.8000    1.2000    0.6000
    0.2000    0.4000    0.6000    0.8000

>> I = Ainv*A;
>> format short e
>> disp(I)
   1.0000e+00  -1.1102e-16            0            0
            0   1.0000e+00            0            0
            0            0   1.0000e+00   2.2204e-16
            0            0            0   1.0000e+00
```

在计算Ainv和I时，由于舍入误差的影响，I(1, 2)和I(3, 4)的值是很小的非零数。

虽然可以使用inv命令求解类似于$Ax=b$的方程组，但是我们偏向于使用第8章介绍的方法来求解，因为它们更加高效而且引入的舍入误差更小。在实际应用中，用$x=A^{-1}b$求解方程$Ax=b$是一种效果很差的办法。

348

7.4.4 对称矩阵

对称矩阵是等于自身转置矩阵的方阵。换句话说，当且仅当$A=A^T$，方阵A是对称矩阵。例如，下列矩阵是对称矩阵：

$$\begin{bmatrix} 5 & -2 & -1 \\ -2 & 6 & -1 \\ -1 & -1 & 3 \end{bmatrix}$$

用矩阵乘以它的转置矩阵后就能得到对称矩阵，对于任意矩阵A，A^TA是对称矩阵。参照下面的例子：

```
>> A = [1:4; 5:8; 9:12]     %  A is *not* symmetric
>> A'*A
ans =
   107   122   137   152
   122   140   158   176
   137   158   179   200
   152   176   200   224

>> A*A'
ans =
    30    70   110
    70   174   278
   110   278   446
```

虽然用A^TA和AA^T产生了两个对称矩阵，但是得到的结果矩阵中元素值的数量级都比原矩阵中元素值数量级大很多，这在数据的最小二乘曲线拟合中是非常重要的（参照第9章）。

7.4.5 三对角矩阵

三对角矩阵（*tridiagonal matrix*）是指除主对角线（$i=j$）、主对角线上方（$j=i+1$）、主对角线下方（$j=i-1$）的值非零外，其他元素都是零的方阵。三对角矩阵出现在许多实际问题中，包括样条插值以及对一维偏微分方程的数值近似。如下例所示：

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}$$

349

通常情况下，不要求每条对角线上的元素值相等。任意一个三对角矩阵可写成

$$A=\begin{bmatrix} a_1 & b_1 & & & & \\ c_2 & a_2 & b_2 & & & \\ & c_3 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & & c_n & a_n \end{bmatrix} \tag{7-31}$$

这说明，矩阵中的元素可以存放在三个向量中，分别是$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$。这样做，我们就把一个整体矩阵分解成了三个相关的数据结构，需要重新考虑它们对加法、减法和乘法的实现。三对角矩阵是稀疏矩阵(sparse matrix)的常见类型（参照附录B）。

例7.8　在MATLAB中创建一个三对角矩阵

用diag函数能够生成对角线上元素的值相等的三对角矩阵，但是要使用diag函数带两个参数的形式。例如，用下列程序可以得到前面介绍过的（-1，2，-1）三对角矩阵：

```
>> A = diag(2*ones(4,1)) - diag(ones(3,1),1) - diag(ones(3,1),-1)
A =
     2    -1     0     0
    -1     2    -1     0
     0    -1     2    -1
     0     0    -1     2
```

也可以由任意的向量a、b、c通过使用函数diag来得到三对角矩阵，如：

```
>> a = ...              % assign values to a, b, and c vectors
>> b = ...
>> c = ...
>> n = length(a);       % must also equal length(b) and length(c)
>>
>> A = diag(a) + diag(b(1:n-1),1) + diag(c(2:n),-1);
```

得到方程（7-31）的矩阵$\boldsymbol{A}$。

NMM工具箱中的函数tridiag可以创建那些对角线上的标量元素重复的三对角矩阵（如，上面的（-1、2、-1）矩阵）或者可以创建由三个向量指定对角线元素的三对角矩阵。 350

7.4.6　正定矩阵

当矩阵所有特征值都是正数时，称这一矩阵为正定矩阵（*positive definite matrix*）。有些矩阵必定为正定矩阵，如，对角占优（diagonally dominant）的对称矩阵。当满足下列情况时，矩阵严格对角占优

$$|a_{ii}| > \sum_{j\neq i}^{n}|a_{ij}|，对所有i=1,\ldots,n \tag{7-32}$$

源于对泊松方程和拉普拉斯方程的保守数值近似得到的矩阵是对称矩阵且是正定矩阵。这些方程出现在导体传热模型、污染物质的传播以及电阻器和电压源组成电路的电流模型中。同时，在固体压力分析、最优化问题的准牛顿法以及多元统计数据的主成分分析中都会产生正定矩阵。

7.4.7 正交矩阵

正交矩阵的各列向量是标准正交集，常用符号Q表示正交矩阵。由于正交矩阵各列正交，所以$Q'Q = I$并且$Q^{-1} = Q'$。

对下面的列向量：

```
>> r = (1/2)*[1 1 1 1]';
>> s = (1/2)*[1 1 -1 -1]';
>> t = (1/2)*[sqrt(2) -sqrt(2) 0 0]';
>> u = (1/2)*[0 0 sqrt(2) -sqrt(2)]';
```

要求读者自己验证这些向量是否是正交向量（利用MATLAB或手工计算）。那么

```
>> Q = [r s t u]
Q =
    0.5000    0.5000    0.7071         0
    0.5000    0.5000   -0.7071         0
    0.5000   -0.5000         0    0.7071
    0.5000   -0.5000         0   -0.7071
>> Q'*Q
ans =
    1.0000         0         0         0
         0    1.0000         0         0
         0         0    1.0000         0
         0         0         0    1.0000
```

351

7.4.8 置换矩阵

置换矩阵（*permutation matrix*）用符号P表示，通过交换单位矩阵的行来得到。乘积$B = PA$的结果B类似于矩阵A，不同的就是A、B中有两行相当于进行了交换操作，这里P是置换矩阵，A是与之匹配的矩阵。例如，3 × 3矩阵

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

当P左乘一个$3 \times n$矩阵时，得到的结果是把原来的矩阵中第二行与第三行交换，代码如下：

```
>> A = [1 2 3 4; 5 6 7 8; 9 10 11 12];  P = [1 0 0; 0 0 1; 0 1 0];
>> B = P*A
B =
     1     2     3     4
     9    10    11    12
     5     6     7     8
```

当P右乘一个$m \times 3$矩阵时，交换对应的列：

```
>> C = A';
>> D = C*P
D =
     1     9     5
     2    10     6
     3    11     7
     4    12     8
```

置换矩阵很容易就可以构造出来。要交换某个矩阵的第r行和第s行，只要把相同大小的单

位矩阵的第r行和第s行交换，然后将所得单位矩阵左乘要置换的矩阵就能得到所要的置换矩阵。注意：置换矩阵也是正交矩阵。

在数学公式中，与置换矩阵相乘能够明确地表示出矩阵中行或列的交换操作，但是在计算机编程时，由于创建矩阵$\boldsymbol{P}$和计算$\boldsymbol{PA}$时都很低效，所以很少使用置换矩阵。若要交换矩阵中行或列，用下列MATLAB语句可以实现对任意矩阵$\boldsymbol{A}$的行或列的交换： [352]

```
>> i1 = ...;    i2 = ...;       % Indices of rows to swap
>> tmp = A(i1,:);               % Save row i1 in the tmp vector
>> A(i1,:) = A(i2,:);           % Copy row i2 into row i1
>> A(i2,:) = tmp;               % move old contents of row i1 into row i2
```

若需要交换矩阵的几行而且不打算使用置换矩阵，那么可以使用置换向量。置换向量在MATLAB的赋值操作中支持行下标。如下例：

```
>> A = [1 1 1; 2 2 2; 3 3 3; 4 4 4; 5 5 5];
>> p = [2 5 3 1 4]';   % length (p) must equal number of rows in A
>> B = A(p,:)          % Copy rows of A into B according to order in p
B =
     2     2     2
     5     5     5
     3     3     3
     1     1     1
     4     4     4
```

表达式A（p，：）是3.5.3节介绍的数组下标的一个应用。给定一个置换向量p，用下列程序可以生成对应的置换矩阵：

```
>> [m,n] = size(A);    % A is target matrix to have it's rows swapped
>> P = eye(m,m);       % P is square
>> P = P(p,:);         % permute the rows of the identity matrix
```

置换矩阵的使用问题再一次说明了：矩阵和向量操作简化了数学描述，但用计算机实现时，这不是一种直接的描述。

7.5 小结

本章复习了线性代数的内容并介绍了作用于向量和矩阵基本操作的MATLAB函数，同时提供了许多信息。要进行线性代数的运算，首先要能够对矩阵和向量进行计算。读完本章内容，要求读者掌握以下内容的手工计算： [353]

- 两个向量或两个矩阵的相加和相减
- 向量的内积
- 向量的外积
- 向量的L_1、L_2和L_∞范数
- 矩阵的L_1和L_∞范数
- 矩阵－向量的积、向量－矩阵的积和矩阵－矩阵的积

进行这些计算的先决条件就是要理解向量和矩阵运算中的匹配问题。在表7-4和表7-5中分别列出了这些匹配条件。当然，读者也应知道在MATLAB中是如何进行这些计算的。

读者对矩阵和向量操作的认识应该不只是停留在标量形式的计算公式程度上。关键思想是把矩阵的各列看作向量，在矩阵－向量乘积和矩阵－矩阵乘积中确实用到了这一思想。下一

章的中心思想之一就是求解方程组$Ax=b$时，涉及到由矩阵A的列向量来构造向量b。

读者除了要了解线性代数如何计算外，还要知道它们潜在的数学思想。特别地，读者应熟悉下列内容：

- 用范数来度量向量和矩阵的大小
- 向量集的线性无关性
- 向量集生成空间的维与向量集中向量个数的区别
- 矩阵的列空间和零空间

表7-3总结了本章用到的MATLAB函数，这些函数不具体地表达数值方法，它们展示的是一种思想，或用于自动创建特殊矩阵如tridiag。本章说明了MATLAB的基本特点就是“*”符号与“'”符号的应用[⊖]。掌握不同类型的矩阵和向量的乘积是学好线性代数的基础。“*”运算符可以用于所有向量或矩阵乘积中，MATLAB解释器只识别合法的“*”运算符。同时，系统会自动创建结果变量的类型——标量、向量和矩阵。这样，在MATLAB中通过编写简洁的代码就能够进行许多计算。

表7-3　本章使用的函数。N.A表示函数不在本书中，但是在NMM工具箱中存在

函　数	小　节	描　述
[illegible]	7.2.1	说明向量的范数
[illegible]	7.2.2	在二维空间中旋转向量
[illegible]	N.A	创建一个满三对角矩阵

354

另一个MATLAB符号“:”在对矩阵中行和列操作中是非常有用的，“:”号是进行矩阵操作中常用到的符号。

表7-4　总结对两个向量操作的约束条件

操　作	例　子	条　件	结　果
加	x + y	x和y都是行向量或列向量，且维数相等（行数或列数）	同x和y相同行数或列数的向量
减	x - y	同加法	同加法
内积	x * y	x是行向量，y是列向量，x的列数等于y的行数	一个标量
外积	x * y	x是列向量，y是行向量，且x的行数等于y的列数	一个矩阵

表7-5　总结对两个矩阵操作的约束条件

操　作	例　子	条　件	结　果
加	A + B	A和B都是$m\times n$矩阵，m是矩阵的行数，n是列数	一个$m\times n$矩阵
减	A - B	同加法	同加法
乘	A * B	A和B的内维数相同，即如果A是$m\times r$矩阵，那么B必须是$r\times n$矩阵	若A是$m\times r$矩阵，B是$r\times n$矩阵，那么C = A*B是$m\times n$矩阵

355

⊖ 下一章会增加“\”运算符。

补充读物

有很多专门介绍线性代数的书，其中有些书还用MATLAB代码来提供例子。许多数值分析人员把MATLAB符号运算符作为对矩阵行和列操作的一种标准来用。

Strang在文献[71、72]中通过具体的例子对线性代数做了很好的介绍。Gill等人在文献[28、第1章]中简洁地总结了数学规则并将它们用于求解线性方程组和优化问题。Datta在文献[12]、Watkins在文献[78]中分别在中等难度水平上介绍了线性代数及其应用。Trefethen和Bau在文献[76]中，用40次讲座形式在中等偏上难度水平上介绍了线性代数，它的中心就是开发读者更深的数学直觉能力。Demmel在文献[15]中实现了现今数值线性代数的一些想法，文献[15]是文献[76]的补充。最经典的书要数文献[32]中Golub和Van Loan介绍的内容，但是该文献读起来不是很容易易懂。在文献[69]中，Stewart在更深层次上给出了LU和QR分解的详细过程，同时还有许多其他的好书可供参考。

这些引用的著作可以让读者深入理解线性代数。许多书还提供了MATLAB代码（如文献[12、72]），还有的书广泛的使用了MATLAB符号（如文献[32、69、76]）。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1.（1）分别编写MATLAB语句实现下列要求：

(a) 创建一个行向量t=[0，π/4，π/2，3π/4，π]。

(b) 创建一个行向量u = [0，π/100，2π/100，...99π/100，π]。

(c) 创建一个列向量v = [sin(0)−cos(0)，sin(π/100)−cos(π/100)，...sin(π)−cos(π)]′。

2.（1）假设x和y是包含n个元素的列向量，u和v是包含m个元素的行向量，A是$m \times n$的矩阵，B是$n \times k$的矩阵。判断附表中的所有表达式的结果是标量、向量、矩阵或表达式是非法的。若结果是向量或矩阵，指出结果的行数和列数。若表达式在一定的情况下才合法，说出满足这些情况的m、n和k值。 [356]

表达式	结果	表达式	结果
$x \quad y$		uAv	
$x \quad u$		AB	
$x'y$		BA	
xy'		A^TA	
yv		$A'Ax$	
$x'A$		$I - xy'$	
uA		$xy'A$	
$x'Ax$		$(Ax)'y$	

3.*（1）手工计算$C = AB$，其中

$$A = \begin{bmatrix} 1 & 1 \\ 2 & 3 \end{bmatrix}, \quad B = \begin{bmatrix} 3 & -1 \\ -2 & 1 \end{bmatrix}$$

A和B有什么关系？

4.（1）给定矩阵A、B和向量v

$$A=\begin{bmatrix}2 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 2\end{bmatrix},\quad B=\begin{bmatrix}3/4 & 1/2 & 1/4\\ 1/2 & 1 & 1/2\\ 1/4 & 1/2 & 3/4\end{bmatrix},\quad v=\begin{bmatrix}3\\ 6\\ 3\end{bmatrix}$$

手工计算下列式子，要求列出详细步骤：

(a) Av

(b) v^tA

(c) v^tAv

(d) AB

矩阵A和B有什么特殊的关系吗？

5. (1) 手工计算下列表达式，要求列出详细的计算步骤并指出使用的算法

(a) AB，其中

$$A=\begin{bmatrix}4 & -3\\ 2 & -1\end{bmatrix},\qquad B=\begin{bmatrix}5 & 3\\ 1 & -2\end{bmatrix}$$

(b) AB，其中

$$A=\begin{bmatrix}5 & 1 & 4 & 0\\ -3 & 0 & 3 & -5\end{bmatrix},\quad B=\begin{bmatrix}3 & 4\\ -1 & 0\\ 0 & -3\\ 3 & -1\end{bmatrix}$$

357

(c) BA，其中A与B等于(b)中的A和B.

(d) AB，其中

$$A=\begin{bmatrix}3 & 2 & 1\end{bmatrix},\qquad B=\begin{bmatrix}-1\\ 0\\ 4\end{bmatrix}$$

(e) BA，其中A与B等于(d)中的A和B.

6. (1)对已知矩阵U计算UU的值

$$U=\begin{bmatrix}-1 & -2 & -3 & -4 & -5\\ 0 & 1 & 3 & 6 & 10\\ 0 & 0 & -1 & -4 & -10\\ 0 & 0 & 0 & 1 & 5\\ 0 & 0 & 0 & 0 & -1\end{bmatrix}$$

矩阵U有什么特殊的性质？

7. (2+) 在用抽象算法描述计算步骤时，用矩阵和向量操作相当方便。但是，当用代码实现这些算法时，有些矩阵和向量的操作将不能使用，有的时候，这些操作很高效，有的时候却很低效。在书写的时候，通常不会很精确地执行下列矩阵-矩阵乘积。

(a) 缩放行：$B=DA$，这里D是对角矩阵

(b) 缩放列：$B=AD$，这里D是对角矩阵

(c) 行置换：$B=PA$，这里P是置换矩阵

(d) 列置换：$B=AP$，这里P是置换矩阵

对上面的矩阵乘法分别写出一个更有效的程序，并与自己觉得可取的程序比较浮点操作次数的多少。

8. （1）若u=（1，0）且v=（1，1），那么它们之间夹角是多少度？

9. （1）在工程上的许多科学计算中，通常需要一些规格化的数或比例数。这些数有时是规定了的，有时是任意的。通常把这些数缩放成在0和1之间的任意数。对下列任意的向量，分别考虑它们的缩放比例：

```
>> x = [-7 -2  3  21 13] ;
>> x1 = x/norm(x,1);
>> x2 = x/norm(x,2);
>> xi = x/norm(x,inf);
```

比较向量x1、x2和xi对应元素的大小，并指定每个比例数和"1"的大小关系。

10. （1+）比较MATLAB用下列语句计算范数的效率（注意：tic、norm（u，2）和toc之间是逗号而不是分号）：

358

```
>> u = linspace(1,2,32000);
>> tic, norm(u,2), toc
>> tic, norm(u,1), toc
>> tic, norm(u,inf), toc
```

要求计算三次并给出最后两次的平均计算时间。

11. *（1）方阵的迹记做tr(A)，它定义为矩阵A对角线上元素的和：

$$\mathrm{tr}(\boldsymbol{A})=\sum a_{ii}$$

要求编写一行MATLAB表达式来计算矩阵A的迹，不能使用for...end循环。

12. （1）要求编写一行MATLAB表达式来计算矩阵A对角线元素的积。（提示：使用内置函数prod。）

13. （1+）在7.2.2节介绍了矩阵-向量按列乘与按行乘的方法。若A是3×4矩阵，x是4×1向量，按列乘得到

```
[illegible]
```

按行乘得到

```
[illegible]
```

为什么按行乘的算法中需要方括号[]，而按列乘时不需要？要求写出按列乘的带方括号的语句。

14. *（1+）用算法7.1编写一个m文件函数（称为matVecCol）计算矩阵-向量（向量在乘号右边）的乘积，要求只能有一个for...end循环。把算法7.1中的内循环用一行带冒号运算符的MATLAB代码替换。随机地输入一对匹配的矩阵A和向量x来测试函数的正确性。

15. （1+）同上题，要求把算法7.2中的内循环用一行带冒号运算符的MATLAB代码替换。

16. （1+）用算法7.3编写一个m文件函数（称为vecMatRow）计算向量-矩阵（向量在乘号左边）的乘积，要求只能有一个for...end循环。把算法7.3中的内循环用一行带冒号运算符的MATLAB代码替换。随机地输入一对匹配的矩阵A和向量x来测试函数的正确性。

[359]

17. （1+）同上题，要求把算法7.4中的内循环用一行带冒号运算符的MATLAB代码替换。

18. （1+）编写函数mymult计算矩阵$\boldsymbol{A}$和矩阵$\boldsymbol{B}$的乘积，并返回结果矩阵$\boldsymbol{C}$。函数的定义要求如下

```
function C = mymult(A, B)
```

要求函数检测输入矩阵是否匹配，若不匹配，打印错误信息，要求函数能够处理下列情况：

(a) $\boldsymbol{A}$是$m \times k$矩阵、$\boldsymbol{B}$是$k \times n$矩阵

(b) 矩阵-向量相乘中的向量右乘（即$\boldsymbol{Ax}$、$\boldsymbol{x}$是一个列向量）

(c) 向量-矩阵相乘中的向量左乘（即$\boldsymbol{x}^T\boldsymbol{A}$、$\boldsymbol{x}$是一个列向量）。

要求不能使用MATLAB提供的内置函数，并且用for循环实现这些操作，通过一个简要的测试来证明所写mymult函数工作正常。

19. （1+）用一个正交矩阵乘以一个向量后不改变向量的L_2范数。几何上，这说明用正交矩阵乘以一个向量后向量的长度不变，它只是按某个数轴旋转或反射。手工计算证明，对任意的α、β和γ，例7.5中三个旋转矩阵都是正交的。

20. （1+）测量例7.6中求矩阵$\boldsymbol{C}$的几种方法的浮点操作次数。矩阵$\boldsymbol{C}_1$和$\boldsymbol{C}_2$分别是：

$$\boldsymbol{C}_1 = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 2 & 2 & \cdots & 2 \\ \vdots & \vdots & & \vdots \\ 128 & 128 & \cdots & 128 \end{bmatrix}, \quad \boldsymbol{C}_2 = \begin{bmatrix} 1 & 2 & \cdots & 128 \\ 1 & 2 & \cdots & 128 \\ \vdots & \vdots & & \vdots \\ 1 & 2 & \cdots & 128 \end{bmatrix}$$

21. * （2）用一个for...end循环编写函数rowScale来缩放矩阵的对角元素。函数定义如下：

```
function B = rowScale(A, d)
```

其中A是被缩放的矩阵，d是缩放因子，它是一个向量。（参照例7.6。）用冒号通配符有助于实现单个循环。计算之前，首先检测并保证矩阵A和向量d是匹配的。

22. （2）同上题，要求编写colScale函数来对矩阵列进行缩放。

23. （2）给定$n \times n$矩阵$\boldsymbol{A}$和n维向量$\boldsymbol{x}$，执行$\boldsymbol{Ax}$计算和$\boldsymbol{x}^T\boldsymbol{A}$计算需要多少次浮点操作？不能只提供结果数据，要求写出详细的计算过程。

24. （1）要求写出方程（7-17）中线性无关条件的矩阵方程（矩阵-向量的积）。

25. * （1）矩阵中各列由方程（7-18）定义。用函数rank来判断矩阵中线性无关的列向量有多少个。

[360]

26. （1）要求同上题，把下列向量作为矩阵的行：

$$\boldsymbol{a} = [1 \quad -3 \quad 4 \quad 2], \quad \boldsymbol{b} = [2 \quad 0 \quad 0 \quad 1]$$

$$\boldsymbol{c} = [-5 \quad 7 \quad 3 \quad 1], \quad \boldsymbol{d} = [-4 \quad -6 \quad 8 \quad 1]$$

27. （1）要求同上题。把向量$\boldsymbol{d}$替换成$\boldsymbol{e}=\boldsymbol{d}-1$。

28. （2）根据7.3节介绍的画图的约定，画出$\boldsymbol{u}$、$\boldsymbol{v}$和$\boldsymbol{w}$的向量图，它们分别定义为

$$\boldsymbol{u} = \begin{bmatrix} 3 \\ -1 \\ 1 \end{bmatrix}, \quad \boldsymbol{v} = \begin{bmatrix} -6 \\ 2 \\ -2 \end{bmatrix}, \quad 且 \; \boldsymbol{w} = \begin{bmatrix} 4 \\ 1 \\ 0 \end{bmatrix}$$

$\boldsymbol{u}$和$\boldsymbol{v}$生成的$\mathbf{R}^3$子空间是什么？$\boldsymbol{u}$、$\boldsymbol{v}$和$\boldsymbol{w}$生成的子空间是什么？

29. (2) 给定向量集{$\boldsymbol{a}$、$\boldsymbol{b}$、$\boldsymbol{c}$、$\boldsymbol{d}$}，其中，$\boldsymbol{a}=(1, 0, 0)^t$、$\boldsymbol{b}=(0, 1, 0)^t$、$\boldsymbol{c}=(0, 0, 1)^t$、$\boldsymbol{d}=2\boldsymbol{a}+\boldsymbol{b}$。给出包含$\boldsymbol{d}$的向量集，并要求这个向量集构成向量集{$\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}$}生成空间的基。向量集中除了$\boldsymbol{d}$以外，还必须有什么向量？为什么？

30. (2) 证明方程（7-21）中的向量构成了$\mathbf{R}^4$的基。

31. (3) 一个向量同它自己的外积生成一个矩阵，这个矩阵可以看作投影算子（projection operator）（参照文献[71，第3章]和文献[76第2章]）。给定任意一个列向量$\boldsymbol{s}$，矩阵

$$\boldsymbol{A} = \boldsymbol{s}\boldsymbol{s}^t$$

是一个投影算子，用一个列向量乘它后得到一个与$\boldsymbol{s}$平行的向量$\boldsymbol{t}=\boldsymbol{A}\boldsymbol{u}$。把$\boldsymbol{A}=\boldsymbol{s}\boldsymbol{s}^t$代入$\boldsymbol{t}$表达式中得到：

$$\boldsymbol{t} = \boldsymbol{A}\boldsymbol{u} = (\boldsymbol{s}\boldsymbol{s}^t)\,\boldsymbol{u} = \boldsymbol{s}\ (\boldsymbol{s}^t\boldsymbol{u})$$

表达式最右边是一个标量$\boldsymbol{s}^t\boldsymbol{u}$乘$\boldsymbol{s}$，所以$\boldsymbol{t}$与$\boldsymbol{s}$在同一个方向上。要求：

(a) 编写一段MATLAB程序证明上面的命题。至少构造两个非平凡向量$\boldsymbol{s}$来计算矩阵$\boldsymbol{A}$。把矩阵$\boldsymbol{A}$应用于另一个非平凡向量$\boldsymbol{u}$，并证明结果向量$\boldsymbol{t}=\boldsymbol{A}\boldsymbol{u}$与向量$\boldsymbol{s}$平行。验证过程的最后的步骤不要通过观察来判断，而要用另一个计算步骤来实现。

(b) 假设$\boldsymbol{s}$是正交基向量$\boldsymbol{e}_1$、$\boldsymbol{e}_2$、…中的一个，其中$\boldsymbol{e}_1 = (1, 0, 0, \ldots)$、$\boldsymbol{e}_2 = (0, 1, 0, \ldots)$矩阵$\boldsymbol{A} = \boldsymbol{e}_i\boldsymbol{e}_i^t$是什么矩阵？这个投影算子对向量会产生什么作用？

32. (2) 给定矩阵

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

求出满足$\boldsymbol{A}\boldsymbol{J} = \boldsymbol{A}$和$\boldsymbol{K}\boldsymbol{A} = \boldsymbol{A}$的矩阵$\boldsymbol{J}$和$\boldsymbol{K}$。

361

33. (2+) 用算法7.5和图7-8证明$\boldsymbol{A}\boldsymbol{I} = \boldsymbol{A}$。

34. (2) 用余子式展开法证明方程（7-29）。

35. (2) 用余子式展开法证明方程（7-30）。

36. (2) 用函数`prod`编写一行MATLAB表达式计算det($\boldsymbol{A}$)，其中$\boldsymbol{A}$是三对角矩阵或三角矩阵。算出浮点操作次数并与运用函数`det`求解时的浮点操作次数作比较。

37. (2+) 在7.3.5节中提到，当`A=tril(1000*ones(128, 128))`时，计算$\boldsymbol{A}$的行列式会产生溢出。解释det($\boldsymbol{A}$)的结果。这个结果正确吗？矩阵$\boldsymbol{A}$是否可逆？

38. (2+) 编写函数判断矩阵是否是对角占优矩阵（参照方程（7-32））。输入变量是一个矩阵，若矩阵是对角占优矩阵，输出1，否则，输出0。用（-1, 2, -1）三对角矩阵检验函数的正确性。

39. (3) 给定矩阵

$$\boldsymbol{A} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}, \quad \text{且}\ \boldsymbol{B} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \sin(\theta) & -\cos(\theta) \end{bmatrix}$$

(a) 手工计算$\boldsymbol{A}^t\boldsymbol{A}$和$\boldsymbol{B}^t\boldsymbol{B}$。$\boldsymbol{A}$和$\boldsymbol{B}$是什么类型的矩阵？

(b) 当$\theta=\pi/4$，$\boldsymbol{u}=[1, 0]^t$、$\boldsymbol{u} = [1, 1]^t$和$\boldsymbol{u} = [0, 1]^t$时，通过计算$\boldsymbol{A}\boldsymbol{u}$和$\boldsymbol{B}\boldsymbol{u}$来说明它们

有什么几何意义？（提示：在（x、y）平面上画出这些向量，向量的尾部在（0，0），顶部在（u_1、u_2），添加一条穿过（0，0）点和（x，y）=（cos（$\theta/2$）、sin（$\theta/2$））点的直线）。NMM工具箱的util目录下的myArrow函数能够自动画图。

40. （2）通过计算证明7.4.7节中的向量$\boldsymbol{r}$、$\boldsymbol{s}$、$\boldsymbol{t}$和$\boldsymbol{u}$都是标准正交向量。

41. （2）假设$\boldsymbol{Q}$是标准正交向量，$\boldsymbol{Q}x$保持了x的L_2范数不变：

$$\|\boldsymbol{Q}x\|_2=\left[(\boldsymbol{Q}x)^T(\boldsymbol{Q}x)\right]^{1/2}=\left[(x^T\boldsymbol{Q}^T)(\boldsymbol{Q}x)\right]^{1/2}=\left[x^T(\boldsymbol{Q}^T\boldsymbol{Q})x\right]^{1/2}$$

$$=[x^Tx]^{1/2}=\|x\|_2$$

362

编写MATLAB语句检验7.4.7节介绍的矩阵$\boldsymbol{Q}$能够保持4个随机x向量的L_2范数不变。

第8章 解方程组

求解方程组是最广泛的自动计算任务之一。本章主要介绍如何建立方程组、使用MATLAB内置函数求解方程组并对影响计算结果可靠性的数值复杂性因素进行分析。图8-1是本章的组织结构，读者至少应学习三个小节的内容：从“基本概念”到“数值法求解$Ax=b$的局限性”，标题为“分解法”的小节详细介绍了反斜杠运算符的使用。对学有余力或有兴趣的读者，可以深入研究最后一小节中介绍的关于非线性方程组求解的内容。

本章主题

1. **基本概念**

   线性方程组的矩阵表示和有解的条件

2. **高斯消去法**

   介绍了用高斯消去（Gaussian elimination）法求解n元方程组的方法，示范了用反斜杠（backslash）运算符手工计算$n \times n$方程组的方法

3. **数值法求解$Ax=b$的局限性**

   介绍了用内置函数求解方程组$Ax=b$并分析了它的局限性，定义了病态（ill conditioned）方程组的特点，介绍了求解方程组所需的计算代价

4. **分解法**

   求解方程组时，现在使用的方法是对矩阵的函数矩阵分解，本节介绍MATLAB如何使用LU和Cholesky分解法，最后，补充说明了使用反斜杠算符求解方程组的算法。

5. **非线性方程组**

   介绍如何用逐次代换法（successive substitution）和牛顿法求解非线性方程组，并实现了用牛顿法求解非线性方程组的方法

图8-1 第8章的主题

例8.1 泵曲线的模型

离心泵是驱动液体在管道中流动的一种常用设备。图8-2是一个典型的泵曲线图（*pump curve*），即流体速度q与压力h的关系曲线。泵的设计不同，泵曲线的形状也不同。尽管如此，在通常情况下，不论泵如何设计，当反向压位差增加时，泵产生的流体速度减小。从图8-2所示的泵曲线中可以看出，在$q=0$的附近，压位差增加，流体速度反而增加。这是从几种类型泵中观测得到的数据。

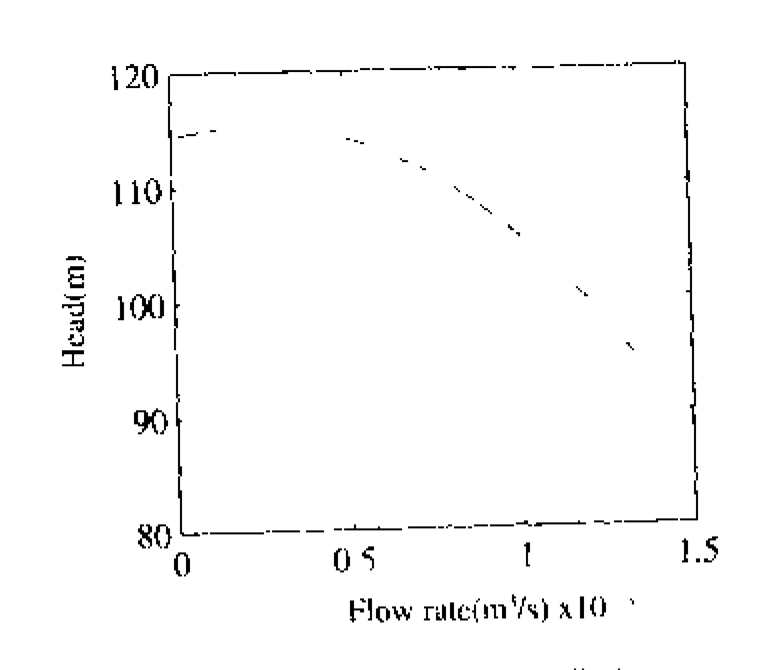


图8-2 离心泵的典型性能曲线

363
↑
364

当要给特定的管道系统选择泵的种类时，就是要找出泵曲线$h_{pump}(q)$与系统曲线$h_{sys}(q)$的交叉点。系统曲线用来描述随着流体通过管道的速度增加，系统中的压力降是如何增加的。泵的制造厂商一般会以图或数据表的形式提供泵曲线。若能以公式的形式提供泵曲线则有助于实现自动计算。图8-2中的$h(q)$曲线与抛物线类似，所以，有必要找出如下的关系式：

$$h=c_1q^2+c_2q+c_3$$

给出的三对数据是（1×10^{-4}，115）、（8×10^{-4}、110）、（1.4×10^{-3}、92.5），它们在图8-2中用空心圆表示。把(q,h)代入以上方程中得到下列三个方程，其中，c_i未知：

$$115 = 1\times10^{-8}c_1+1\times10^{-4}c_2+c_3$$

$$110 = 64\times10^{-8}c_1+8\times10^{-4}c_2+c_3$$

$$92.5 = 196\times10^{-8}c_1+14\times10^{-4}c_2+c_3$$

上面的方程组可以写成如下矩阵形式

$$\begin{bmatrix} 1\times10^{-8} & 1\times10^{-4} & 1 \\ 64\times10^{-8} & 8\times10^{-4} & 1 \\ 196\times10^{-8} & 14\times10^{-4} & 1 \end{bmatrix}\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}=\begin{bmatrix} 115 \\ 110 \\ 92.5 \end{bmatrix}$$

用更简洁的符号表示上面方程组得到$Ax=b$，其中

$$A=\begin{bmatrix} 1\times10^{-8} & 1\times10^{-4} & 1 \\ 64\times10^{-8} & 8\times10^{-4} & 1 \\ 196\times10^{-8} & 14\times10^{-4} & 1 \end{bmatrix},\quad x=\begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix},\quad b=\begin{bmatrix} 115 \\ 110 \\ 92.5 \end{bmatrix}$$

一旦把方程组写成矩阵的形式，用本章介绍的MATLAB函数能够很容易地求得解。

8.1 基本概念

使用计算机自动求解线性方程组之前，要求读者掌握一定的概念和原理。本节介绍如何用矩阵公式求解方程组以及方程组有解的条件。

8.1.1 矩阵公式

要用数值方法求解线性方程组，首先要把方程组写成标准形式。本章中，一般把方程组写成：

365

$$\boldsymbol{Ax} = \boldsymbol{b} \tag{8-1}$$

其中，$\boldsymbol{A}$是$m\times n$的实数矩阵，$\boldsymbol{x}$是$n\times1$未知向量，$\boldsymbol{b}$是一个已知的$m\times1$的实数向量。通常，$m=n$，但这不是必需的。在8.1.2节中提供的方程组就是m不等于n的更一般的方程组形式。但是，本章大部分介绍的都是$m=n$形式的方程组。在第9章中将会介绍$m>n$形式的方程组。

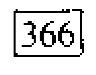
在实际问题求解中，要把问题表示成线性方程组，首先要用符号表示出各个量之间的关系，然后把这些常用符号表示成符合线性代数规范的$\boldsymbol{A}$、$\boldsymbol{x}$和$\boldsymbol{b}$形式。下面是把方程表示转化为标准线性方程组形式的步骤：

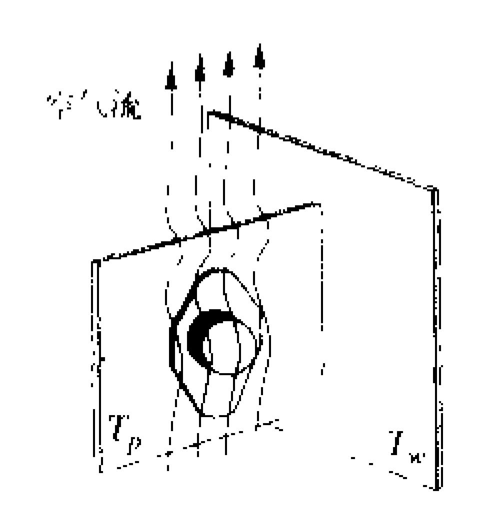
1. 写出方程的自然表达形式
2. 确定未知量，按顺序排列
3. 分离未知量
4. 用矩阵的形式表示方程组

步骤1是为了提醒读者不要把问题直接表示成矩阵公式的形式。步骤2到步骤4将自然形式表示的方程组变换成线性代数表示形式。步骤2中，把未知量指定为方程（8-1）中向量$\boldsymbol{x}$的元素。步骤3是对线性方程组的重新排列，对每个方程，把未知量移到等号的左边，剩余项移到右边。然后，把等号左边各项组合，使每个未知量只出现一次。完成了前3个步骤后，第4个步骤就能很容易地完成。

例8.2 IC元器件的冷却

随着电路的时钟速度的增加和在给定空间内所安装晶体管数量的增加，集成电路的速度和性能都极大地提高了。同时，时钟速度和晶体管密度的增加也提高了IC器件的功耗，这些功耗最终会以热量的形式散发出来，如果耗电量太大会导致元器件温度升高。为了使元器件能够正常工作，需要给它做降温处理，本例说明了如何根据IC元件冷却模型的数学形式得出它的线性方程组。

写出方程的自然形式 图8-3a中的插图画出了TO-3封装形式的一个电子组件，这个组件或为功率管，或为其他大功率器件。在本例中IC元件被安装在散热片（heat spreader）上，然后把散热片固定在另一个平板上，这个平板可能是包装该IC元件和其他器件（没有表示出来）的机壳（箱）的内壁。图中T_p是散热片的温度，T_w是平板的温度。 366



周围温度T_a

a) IC元件装在散热片上以加速散热

b) 从IC元件到机箱的热流网模型

图 8-3

IC元件消耗的电能为Q_c（W），它以热量的形式散发出去。因为IC元件的温度T_c比周围空气的温度T_a高，所以IC元件附近的空气因温度升高而上升。空气流动有助于降低IC元件、散热片和平板的温度。IC元件的热量除了传给周围空气，也传给散热片；散热片的热量除了传给周围空气，也传给平板；平板的热量传给周围的空气。图8-3b说明了在这个系统中传热的网络图。网络中的结点是已定义温度参数的发热点，箭头表示传热的方向，网络中的电阻可以看作热敏电阻。传递一定的热量，大电阻需要改变的温度差较高而小电阻需要改变的温度差较小。

把能量守恒定理应用到IC器件和它的周边部件中，得到下列方程：

$$Q_1 = \frac{1}{R_1}(T_c - T_p), \qquad Q_4 = \frac{1}{R_4}(T_p - T_a), \qquad Q_c = Q_1 + Q_3,$$

$$Q_2 = \frac{1}{R_2}(T_p - T_w), \qquad Q_2 = \frac{1}{R_5}(T_w - T_a), \qquad Q_1 = Q_2 + Q_4,$$

$$Q_3 = \frac{1}{R_3}(T_c - T_a)$$

注意，在第二行中间的方程中用到了$Q_5 = Q_2$，因为在节点T_w上只有一个输入和输出热流的路径。 367

确定未知量 在本例的分析中，晶体管Q_c散发的热量和环境的温度T_a都是已知的。电阻能

够由不同物理配置中的材料特性、物理尺寸和热流试验关系计算得到（参见F.P. Incorpera和D.P. Dewitt，*Fundamentals of Heat and Mass Transfer*，4th edition，1996，John Wiley &Sons，New York）。因此，Q_c、T_a和所有的R_i都是已知量。未知量是Q_1、Q_2、Q_3、Q_4、T_c、T_p和T_w。根据方程（8-1）得到：

$$x=[Q_1,\ Q_2,\ Q_3,\ Q_4,\ T_c,\ T_p,\ T_w]^t$$

有7个方程和7个未知量，所以可以找到关于x的解。

分离未知量　把上面的方程组重新排列得到：

$$\begin{aligned}
&R_1Q_1-T_c+T_p=0\\
&R_2Q_2-T_p+T_w=0\\
&R_3Q_3-T_c=-T_a\\
&R_4Q_4-T_p=-T_a\\
&R_5Q_2-T_w=-T_a\\
&Q_1+Q_3=Q_c\\
&Q_1-Q_2-Q_4=0
\end{aligned}$$

注意到，在每个方程中，向量x中的未知量都出现在等号的左边，已知量都出现在等号的右边。

用矩阵的形式表示方程组　上面方程组的矩阵形式如下：

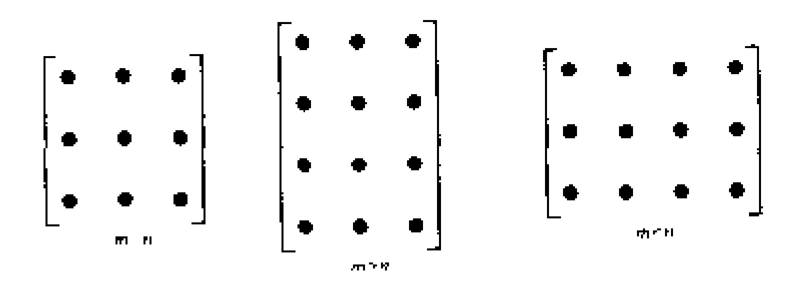
$$\begin{bmatrix}R_1&0&0&0&-1&1&0\\0&R_2&0&0&0&-1&1\\0&0&R_3&0&-1&0&0\\0&0&0&R_4&0&-1&0\\0&R_5&0&0&0&0&-1\\1&0&1&0&0&0&0\\1&-1&0&-1&0&0&0\end{bmatrix}\begin{bmatrix}Q_1\\Q_2\\Q_3\\Q_4\\T_c\\T_p\\T_w\end{bmatrix}=\begin{bmatrix}0\\0\\-T_a\\-T_a\\-T_a\\Q_c\\0\end{bmatrix}\tag{8-2}$$

给定T_a、Q_c和所有的R_i，求解这个含7个未知量的方程组可以求出热流和温度。

8.1.2　方程组有解的条件

方程（8-1）中解的性质依赖于矩阵A的形状、b是否在A的列空间中和A中各列是否线性无关。参照7.3节中相关背景材料的讨论，这有利于理解下面的内容。

通常，矩阵有m行和n列，它的每行对应线性方程组左边未知量的系数，每列对应x的一个元素。[368] 矩阵方程$Ax=b$中，A是$m\times n$矩阵，说明有m个方程、n个未知量。图8-4根据m与n的大小关系



求$Ax=b$关于n个方程的n个未知量　　超定方程组（如：最小二乘问题）　　不定方程组（如：最优化）

图8-4　m行n列矩阵的形状和应用。“·”号表示矩阵中的元素

列出了三种情况。比较普遍的情况（大部分人认为只有这一种情况）是有n个方程和n个未知量。当$m>n$时，方程数多于未知量的个数，称这个方程组是超定方程组(overdetermined)，通常超定方程组用最小二乘法求解，它应用在把m个（x，y）数据对拟合为有n个系数的方程（参照第9章）。当$m<n$时，方程的个数少于未知量个数，称这种方程组是不定方程组（underdetermined），这种方程组要么没有解，要么会有无数个解，它常用于数值优化中（参照文献[28，72]）。

相容性 方程（8-1）中向量$\boldsymbol{x}$的元素作为矩阵$\boldsymbol{A}$中各列向量线性组合的系数，即：

$$\boldsymbol{Ax}=\boldsymbol{b} \Leftrightarrow x_1\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix}+x_2\begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix}+\cdots+x_n\begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix}=\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{8-3}$$

369

方程（8-1）存在关于$\boldsymbol{x}$的解则要求方程（8-3）中的线性组合是有可能的。根据向量空间的概念（参照7.3.2节），方程（8-1）有解当且仅当向量$\boldsymbol{b}$在矩阵$\boldsymbol{A}$的列空间中。这种方程组称为相容方程组(consistent)[⊖]。若$\boldsymbol{b}$不在$\boldsymbol{A}$的列空间中，那么$\boldsymbol{A}$的列向量无法线性组合得到向量$\boldsymbol{b}$，称这种方程组是不相容的(inconsistent)。不相容方程组应用在许多方面，如例8.3和第9章的例子。当$\boldsymbol{A}$是$n \times n$矩阵，且$\boldsymbol{A}$非奇异时能够保证方程组的相容性。

构造增广矩阵（augmented matrix）$\tilde{\boldsymbol{A}}$能够简明地表示出$\boldsymbol{A}$和$\boldsymbol{b}$的相容性。由矩阵$\boldsymbol{A}$和向量$\boldsymbol{b}$能够得到增广矩阵$\tilde{\boldsymbol{A}}$：

$$\tilde{\boldsymbol{A}}=\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}$$

$\boldsymbol{A}$和$\boldsymbol{b}$相容当且仅当$\tilde{\boldsymbol{A}}$和$\boldsymbol{A}$的秩相同。当它们的秩相同时，$\tilde{\boldsymbol{A}}$的列向量线性相关，且$\boldsymbol{b}$向量在$\boldsymbol{A}$的列向量生成的子空间中。

例8.3 数据拟合中的不相容方程组

假设通过实验得到下列$y=f(x)$数据：

x	1	2	3
y	2	1	0.5

图8-5画出了这些数据点，它们看似很接近，但是不完全在同一条直线上。由实验结果可以采用如下应用模型：

$$y=\alpha x+\beta$$

这是一个直线方程。把实验数据代入上式并把方程写成矩阵形式后得到：

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix}\begin{bmatrix} \alpha \\ \beta \end{bmatrix}=\begin{bmatrix} 2 \\ 1 \\ 0.5 \end{bmatrix}$$

⊖ 有些作者用“兼容”（如文献[28]）。

矩阵中各列向量线性无关，但方程个数多于未知量个数。能否得到这个方程的解？这要看系数矩阵和等号右边的向量是否相容，

370

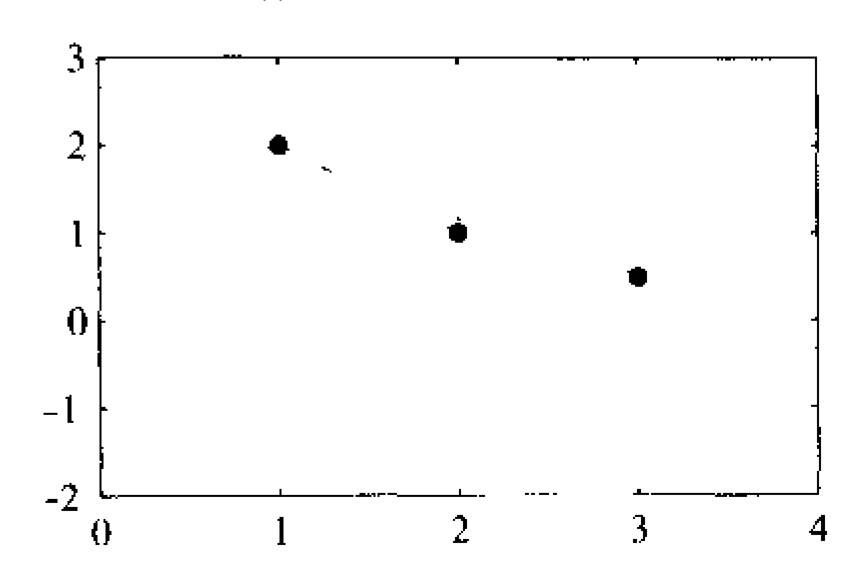


图8-5 把三对数据点（实心圆表示）拟合成一条直线得到不相容方程组

注：由于这三个不在同一直线上的点不能用一个带两个参数的线性函数表示，所以方程组是不相容的，用最小二乘法能够得到较精确的解（用实线表示）。若把y值改成0（由实心圆点所在的位置移到了空心圆点所在的位置），那么三个数据点都在虚线上。虚线上的数据点对应一个相容的方程组，它有三个方程和两个未知量。

```
>> A = [1 1; 2 1; 3 1];   b = [2; 1; 0.5];
>> rank(A)
ans =
     2

>> rank([A b])
ans =
     3
```

由于rank(***A***) < rank([***A b***])，所以向量***b***不在***A***的列空间中，不能求出α和β的精确解。图8-5也说明了这点。数据点不处于同一条直线上，所以没有直线方程能够同时经过这三点。但是，用最小二乘法能够拟合一条与这三点距离最近的直线，并求出直线方程。最小二乘法将在第9章详细介绍。

当把第三组数据中的y值由0.5改成0，其他两个数据对都不变，得到关于α、β的方程组如下所示：

$$\begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$$

371

相容性测试后得出结论是，这个方程组存在精确解。

```
>> A = [1 1; 2 1; 3 1];   b = [2; 1; 0];
>> rank(A)
ans =
     2

>> rank([A b])
ans =
     2
```

等号右边的向量***b***在矩阵***A***的列空间中。尽管上面的方程组有三个方程和两个未知量，方程组仍然有解，$[\alpha, \beta]^T=[-1, 3]^T$。$\alpha=-1$、$\beta=3$的直线经过三个数据点，如图8-5中虚线所示。

矩阵-向量按列相乘时，第二个方程组的解可以写成

$$\begin{bmatrix}2\\1\\0\end{bmatrix}=(-1)\begin{bmatrix}1\\2\\3\end{bmatrix}+(3)\begin{bmatrix}1\\1\\1\end{bmatrix}$$

对于$m \times n$的方程组，并非向量$\boldsymbol{b}$都在矩阵$\boldsymbol{A}$的列空间中，即使数据发生一点点变化，如把$y_1=0$换成$y_1=0.000001$，也会产生完全不同的结果，导致方程组得不到解。

练习11和14说明了本例中用高斯消去法求解的过程及结论。

rank(A)的作用　矩阵$\boldsymbol{A}$与向量$\boldsymbol{b}$的相容性可用来判断方程（8-1）是否存在精确解，另一个也很重要的问题是，该方程组是否只有惟一解？如何判断方程只有惟一解？若存在惟一解，可以用矩阵$\boldsymbol{A}$的秩来判断。

7.3.1节中提到当$\boldsymbol{A}$的各列向量线性相关时，$\boldsymbol{A}z=0$存在非平凡解（nontrivial solution）。假设$\boldsymbol{A}$的各列向量线性相关，那么存在非零解z。若$\boldsymbol{x}$是$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$的解，那么$\boldsymbol{x}+\sigma z$也是方程的解，这是因为

$$\boldsymbol{A}(\boldsymbol{x}+\sigma z)=\boldsymbol{A}\boldsymbol{x}+\sigma \boldsymbol{A}z=\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$$

所以，如果矩阵$\boldsymbol{A}$的各列向量线性相关，那么，方程只要有解，解就不是惟一的。

当矩阵$\boldsymbol{A}$的各列向量线性无关时，假设$\tilde{x}$和$\hat{x}$都是方程$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$的解。将$\boldsymbol{A}\tilde{x}=\boldsymbol{b}$减去$\boldsymbol{A}\hat{x}=\boldsymbol{b}$得到

$$0=\boldsymbol{A}\tilde{x}-\boldsymbol{A}\hat{x}=\boldsymbol{A}(\tilde{x}-\hat{x})$$

由于$\boldsymbol{A}$是线性的（不依赖于x），所以可以进行第二步操作$\boldsymbol{A}\tilde{x}-\boldsymbol{A}\hat{x}=\boldsymbol{A}(\tilde{x}-\hat{x})$。已经假设矩阵$\boldsymbol{A}$的各列向量线性无关，$\boldsymbol{A}(\tilde{x}-\hat{x})=0$的解是$\tilde{x}-\hat{x}=0$，即$\tilde{x}=\hat{x}$，所以，方程的解惟一。 372

根据7.3.4节的内容知rank($\boldsymbol{A}$)等于矩阵$\boldsymbol{A}$中线性无关列的个数。因此，上面一段得到的结论可以写成：设$\boldsymbol{A}$是$m \times n$矩阵

- 当rank($\boldsymbol{A}$) $< n$且$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$有解时，方程组有无数解
- 当rank($\boldsymbol{A}$) $= n$且$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$有解时，方程组有惟一解

需要记住，相容性是判断解存在与否的依据。

对于$n \times n$方程组，相容性条件由矩阵$\boldsymbol{A}$的秩来决定。若$\boldsymbol{A}$是$n \times n$矩阵，且秩为n，那么它的列空间是$\boldsymbol{R}^n$，因此，所有含n个元素的向量都在矩阵$\boldsymbol{A}$的列空间中。因此

对于含n个方程和n个未知量的方程组$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$，$\boldsymbol{x}$的解存在且惟一的条件是当且仅当rank($\boldsymbol{A}$)=$n$

相反地，$n \times n$矩阵$\boldsymbol{A}$的秩小于n时，方程组$\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$可能不相容，它或者无解或者存在多个的解（参照例8.4）。

读者要注意：计算矩阵的秩有时会出现一些问题。前面用到rank($\boldsymbol{A}$)都是描述性的，都没有给出计算秩的具体数值计算算法。尽管用矩阵的秩能够很容易地知道矩阵和方程组的特性，但是矩阵的秩通常是不计算的。就是说，读者在做本章习题时会发现用MATLAB内置函数rank计算矩阵的秩很方便，由于练习中的矩阵都很小而且是良态的（参见8.3.2节），所以用rank函数求这些矩阵的秩所得结果是正确的。

当$m=n$时，方程组的形式解　如7.4.3节所说，当$\boldsymbol{A}$是$n \times n$矩阵时，方程（8-1）的形式解（formal solution）为

$$x = A^{-1}b \tag{8-4}$$

A^{-1}是矩阵A的逆。在数学上，方程（8-4）是正确的[⊖]，它直接由方程（8-1）得到。但是，只有单独计算A^{-1}后才能求出x。所以说，形式解只是一种数学表示，不是求解方程组的方法。

373

即使A^{-1}已经求出来（或有计算它的程序），用$x = A^{-1}b$求解方程组仍不是一种好的方法。与这种办法相比用高斯消去法所需的浮点操作次数少，高斯消去法不但能节省计算时间（对小型方程不重要），而且结果受舍入误差的影响更小。方程（8-4）可以看成是"求解关于x的方程组$Ax = b$"的一种简便标记法。矩阵的逆，同行列式（7.3.5节）和rank（A）一样，都是一种概念表示方法，并不适合用于数值计算中。

A^{-1}存在与否（不论它是否真正地计算出来，只是从数学意义上看是否存在）用于判断方程组$Ax = b$是否有精确解。若A^{-1}存在，称A是可逆或非奇异的(nonsingular)；否则，称A不可逆或奇异的(singular)。

总结 方程（8-1）解的特性由方程组的相容性和rank（A）来决定。相容性用于判断方程组是否有解，矩阵A的秩用于判断方程组解的惟一性。对于方程组$Ax = b$，其中，A是$m \times n$矩阵，相容性与秩的含义如下：

- 若rank(A) =n而且方程组相容，那么方程组存在惟一解
- 若rank(A)= n而且方程组不相容，那么方程组没有解
- 若rank(A)< n而且方程组相容，那么方程组有无数解

对于$n \times n$的方程组，若rank(A) =n，则自动保证了方程组是相容的。当A是$n \times n$矩阵时，下列条件互相等价：

- 矩阵A中各列线性无关
- 矩阵A中各行线性无关
- rank(A) = n
- det(A) ≠ 0
- A^{-1}存在
- 方程组$Ax = b$有解且惟一

例8.4 奇异性的几何解释

374

在x-y平面上，两条直线的交点可以用2 × 2方程组求解，如：

$$y = -2x + 6$$

$$y = \frac{1}{2}x + 1$$

由于这两条直线的斜率不同，所以存在一个交点。把以上方程组写成矩阵形式：

$$\begin{bmatrix} 2 & 1 \\ -1/2 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \end{bmatrix} \tag{8-5}$$

上式中$x_1 = x$，$x_2 = y$。方程组的解就是向量$(x_1, x_2)^T = (2, 2)^T$，它是两条直线交点的坐标。交点是同时在这两条直线的惟一点，只有它的值才能同时满足两个方程。

用几何方法能够简单地解释2 × 2方程组有解、无解以及有多个解的情况。改变方程（8-5）中的系数矩阵和等号右边的向量后得到不同的情况，在图8-6中列出了可能出现的情况。图8-6

⊖ 当$m \neq n$时，A^{-1}不存在，方程（8-4）只适用于$n \times n$的方程组。

中左上角的图是方程（8-5）的解，两条直线相交于一点，方程组是相容的，而且矩阵**A**是非奇异的。

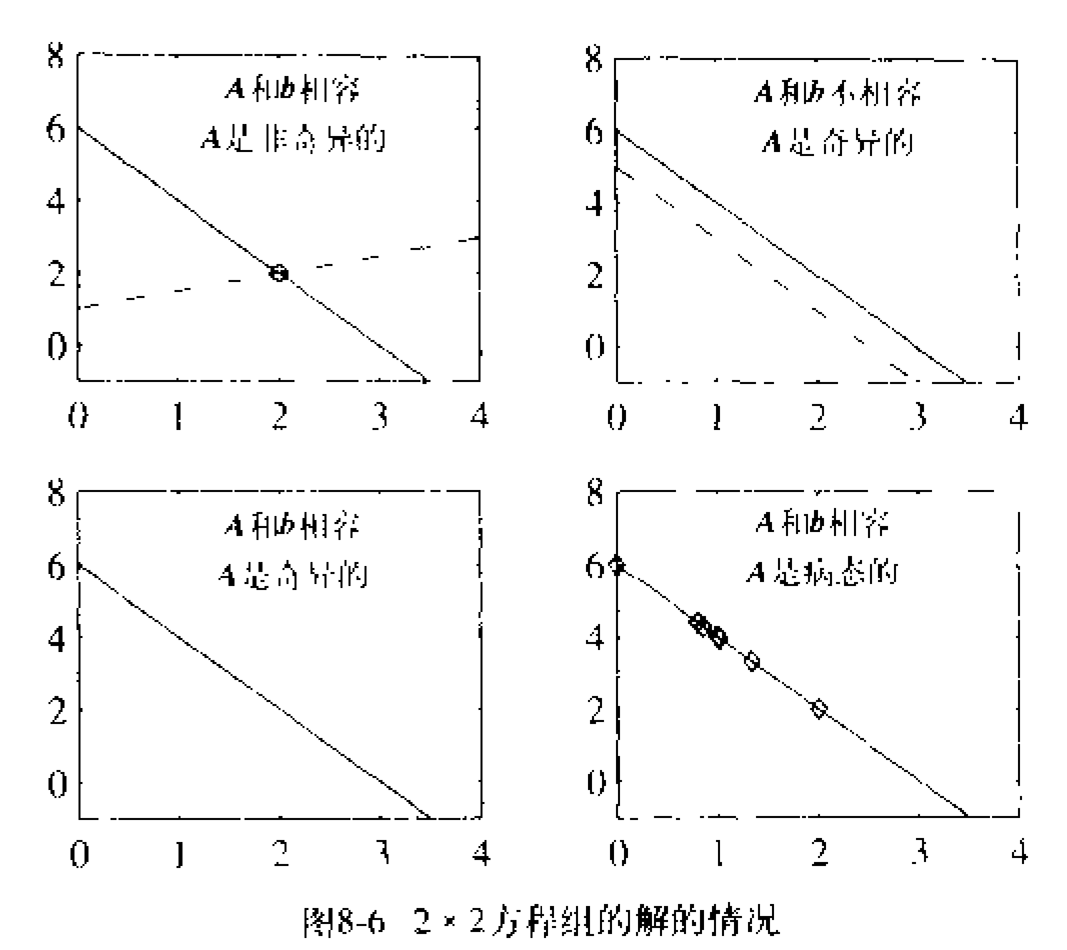


图8-6 2×2方程组的解的情况

注：左上角的图说明方程组存在唯一的解，它就是图中用空心圆画出的交点。右下角的图说明方程组存在多个解，用菱形表示交点，而且不同的菱形表示方程（8-6）中不同的δ值

[375]

图8-6右上角的图对应以下方程组：

$$\begin{bmatrix}2 & 1\\2 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}6\\5\end{bmatrix}$$

两条直线互相平行，所以方程组无解。系数矩阵的行向量线性相关，所以矩阵是奇异的。系数矩阵的列空间是直线$x_2 = x_1$，因为等号右边的向量$[6\ 5]^T$不在这条直线上，所以，方程组是不相容的。

当方程组相容（矩阵**A**和向量**b**）而系数矩阵奇异时，方程组存在无数个解。对于2×2方程组，这种情况下，两条直线重合，如图8-6左下角所示。此时，方程组的矩阵形式是：

$$\begin{bmatrix}2 & 1\\2 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}6\\6\end{bmatrix}$$

注意：系数矩阵的列线性组合后能够得到向量$[6\ 6]^T$。

最后，稍微改变系数矩阵中a_{21}的值和等号右边向量中b_2的值得到：

$$\begin{bmatrix}2 & 1\\2+\delta & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}6\\6+\delta\end{bmatrix} \tag{8-6}$$

当δ =logspace（-16，-8)时，得到图8-6右下角所示图。每一个解对应不同的δ值，而且每个解在直线$x_2 = -2x_1 + 6$上并用菱形标出。尽管δ值与a_{21}和b_2相比都非常小，但是结果却完全不同，这是病态矩阵的特点：输入（系数矩阵和等号右边的向量）发生很小的变化都会导致结果（解向量）发生很大的变化。8.3.2节将介绍这个问题的修正方法。练习15给出了这个病态方程组的其他形式。

例8.5 惠斯通（Wheatstone）电桥模型

惠斯通电桥是一个电路，它用于放大和测量从传感器送来的信号。图8-7是惠斯通电桥结构中的一种。电路中所接入的电压V_{in}用来产生电阻网络中的电流，电阻R_2是传感器电路中一部分（电路的其他部分没有画出来），所以R_2的值会随着周围环境变化而变化（如温度和形变）。挑选好电桥中其他电阻的阻值大小，能够使得待测量电压V_{out}的值随R_2值的变化而变化。

376

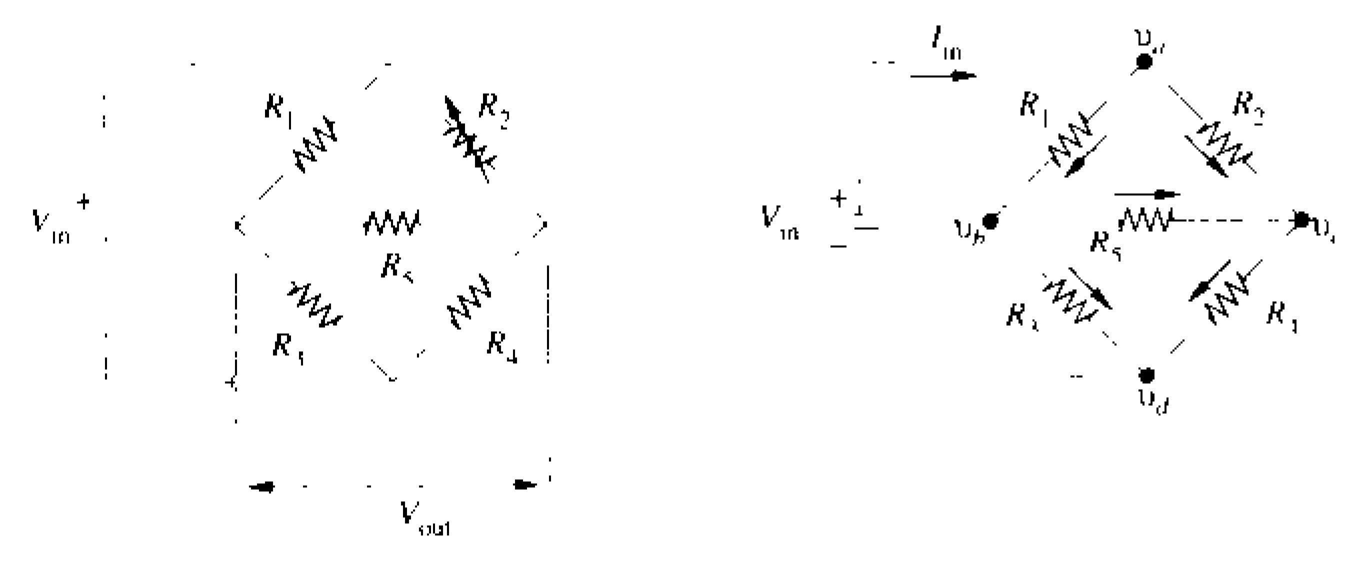


图8-7 惠斯通电桥的电路图（左边）及节点电压定义（右边）

分析电桥电路后可以得到线性方程组，可以用8.1.1节介绍的四个步骤逐步得出矩阵形式的方程组。按照前面介绍的求解方程组的四个步骤，首先得到一个奇异的系数矩阵，这里解释了奇异产生的原因并提出了补救措施。

用普通符号表示方程组 应用基尔霍夫（Kirchhoff）电压法则（回路中总电压是零）和电流法则（流入和流出节点上的电流代数和是零）。在图8-7的右图中，给每个节点标注了节点电压，两个节点间的箭头表示电流的流向[⊖]。每个电阻中流过的电流分别是

$$i_1=\frac{v_a-v_b}{R_1},\quad i_2=\frac{v_a-v_c}{R_2},\quad i_3=\frac{v_b-v_d}{R_3},\quad i_4=\frac{v_c-v_d}{R_4},\quad i_5=\frac{v_b-v_c}{R_5}$$

根据基尔霍夫定理得到四个节点v_a、v_b、v_c和v_d的电流分别是

$$I_{in}-\frac{v_a-v_b}{R_1}-\frac{v_a-v_c}{R_2}=0 \tag{8-7}$$

$$\frac{v_a-v_b}{R_1}-\frac{v_b-v_c}{R_5}-\frac{v_b-v_d}{R_3}=0 \tag{8-8}$$

$$\frac{v_a-v_c}{R_2}+\frac{v_b-v_c}{R_5}-\frac{v_c-v_d}{R_4}=0 \tag{8-9}$$

$$\frac{v_b-v_d}{R_3}+\frac{v_c-v_d}{R_4}-I_{in}=0 \tag{8-10}$$

确定未知量 这个电路中有5个未知量，它们分别是4个节点电压v_a、v_b、v_c、v_d和电源电流I_{in}。只有4个方程，所以还需要更多的关于电路的信息并得出新的方程。

377

把方程（8-7）和方程（8-10）相加后可以消去I_{in}。方程如下：

⊖ 图8-7中给出的电流方向不一定是实际的电流方向，而是假设的电流方向。若所求电流方向和与现在的方向相反，那么得到的电流是个负数。

$$\frac{v_b - v_d}{R_3} + \frac{v_c - v_d}{R_4} - \frac{v_a - v_b}{R_1} - \frac{v_a - v_c}{R_2} = 0 \tag{8-11}$$

注意到，v_a和v_d的值由电源电压V_{in}决定，又可以得到如下方程：

$$v_a = v_d + V_{in} \tag{8-12}$$

把方程（8-12）代入方程（8-11）（8-8）和（8-9）后得到：

$$\frac{v_b - v_d}{R_3} + \frac{v_c - v_d}{R_4} - \frac{v_d + V_{in} - v_b}{R_1} - \frac{v_d + V_{in} - v_c}{R_2} = 0 \tag{8-13}$$

$$\frac{v_d + V_{in} - v_b}{R_1} - \frac{v_b - v_c}{R_5} - \frac{v_b - v_d}{R_3} = 0 \tag{8-14}$$

$$\frac{v_d + V_{in} - v_c}{R_2} + \frac{v_b - v_c}{R_5} - \frac{v_c - v_d}{R_4} = 0 \tag{8-15}$$

这是含3个方程、3个未知量的方程组，未知向量是$\boldsymbol{x} = (v_b, v_c, v_d)^T$。

分离未知量 把方程（8-13）～（8-15）重新排列并分离出v_b、v_c和v_d得到

$$\left[\frac{1}{R_1} + \frac{1}{R_3} + \frac{1}{R_5}\right] v_b - \frac{1}{R_5} v_c - \left[\frac{1}{R_1} + \frac{1}{R_3}\right] v_d = \frac{1}{R_1} V_{in} \tag{8-16}$$

$$-\frac{1}{R_5} v_b + \left[\frac{1}{R_2} + \frac{1}{R_4} + \frac{1}{R_5}\right] v_c - \left[\frac{1}{R_2} + \frac{1}{R_4}\right] v_d = \frac{1}{R_2} V_{in} \tag{8-17}$$

$$\left[\frac{1}{R_1} + \frac{1}{R_3}\right] v_b + \left[\frac{1}{R_2} + \frac{1}{R_4}\right] v_c - \left[\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}\right] v_d = \left[\frac{1}{R_1} + \frac{1}{R_2}\right] V_{in} \tag{8-18}$$

写成矩阵形式 上面方程组的矩阵形式是

$$\begin{bmatrix} \left[\frac{1}{R_1} + \frac{1}{R_3} + \frac{1}{R_5}\right] & -\frac{1}{R_5} & -\left[\frac{1}{R_1} + \frac{1}{R_3}\right] \\ -\frac{1}{R_5} & \left[\frac{1}{R_2} + \frac{1}{R_4} + \frac{1}{R_5}\right] & -\left[\frac{1}{R_2} + \frac{1}{R_4}\right] \\ \left[\frac{1}{R_1} + \frac{1}{R_3}\right] & \left[\frac{1}{R_2} + \frac{1}{R_4}\right] & -\left[\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4}\right] \end{bmatrix} \begin{bmatrix} v_b \\ v_c \\ v_d \end{bmatrix} = \begin{bmatrix} \frac{1}{R_1} \\ \frac{1}{R_2} \\ \frac{1}{R_1} + \frac{1}{R_2} \end{bmatrix} V_{in}$$

这个方程组没有惟一解，因为左边的3×3系数矩阵是奇异的，第三列是前两列的和，第三行是前两行的和。这种问题的出现不是由公式错误引起，而是由电桥本身的性质决定的。给v_b、v_c、v_d的确定电平值上分别加上任意一个常量，这个电桥仍然起作用。给电桥加上接地线可以避免这种浮动电压（floating potential）问题。若在节点v_d上添加接地线，那么v_d的浮动电压是零。对于矩阵公式，从数学上讲，只要把v_d设为零，就可以消去矩阵的第三列和第三行。 378

$$\begin{bmatrix} \left[\frac{1}{R_1} + \frac{1}{R_3} + \frac{1}{R_5}\right] & -\frac{1}{R_5} \\ -\frac{1}{R_5} & \left[\frac{1}{R_2} + \frac{1}{R_4} + \frac{1}{R_5}\right] \end{bmatrix} \begin{bmatrix} v_b \\ v_c \end{bmatrix} = \begin{bmatrix} \frac{1}{R_1} \\ \frac{1}{R_2} \end{bmatrix} V_{in} \tag{8-19}$$

课后习题24到26将深入研究这个例子。

本例旨在告诉读者，对实际的模型建立的方程组有可能是奇异的，这时，需要应用该实际问题的更多的附加信息。

8.2 高斯消去法

向后代入的高斯消去法是用于求解线性方程组的基本方法[⊖]。虽然Strange在文献[71、72]中说明了如何用消去法揭示矩阵（$m \times n$）的内在性质，但高斯消去法常用于方阵的计算中(即$n \times n$矩阵)。若不明确说明、本书后面所有系数矩阵都按方阵处理。

用高斯消去法求解方程组主要包含两个步骤。第一步——消去——对矩阵按行操作、将矩阵转换成上三角矩阵。消去需要较大的工作量，而且也是最容易受舍入误差影响的阶段。第二步是用向后代入法求出方程组的解。在介绍消去法之前，我们来看看为什么首先要把系数矩阵转换成上三角矩阵以及这样做的好处。

8.2.1 解对角方程组

对角矩阵的方程组很容易求解，因为方程组中很明显地包含了方程组的解。如，方程组 379 $Ax = b$、其中

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{bmatrix}, \quad b = \begin{bmatrix} -1 \\ 6 \\ -15 \end{bmatrix}$$

等价于

$$\begin{aligned} x_1 \qquad\qquad &= -1 \\ 3x_2 \qquad &= 6 \\ 5x_3 &= -15 \end{aligned}$$

显然，解是$x_1 = -1$，$x_2 = 2$，$x_3 = -3$。下面是求解对角方程组的算法：

算法8.1　对角方程组的求解

given A、b

for $i = 1 \ldots n$

　　$x_i = b_i / a_{i,i}$

end

在MATLAB中，用下列语句就能求出对角方程组的解：

```
x = b./diag(A)
```

8.2.2 求解三角方程组

在上三角矩阵中，主对角线下方的元素都是零；下三角矩阵中，在主对角线上方的元素都是零。它们的一般形式分别是：

⊖ 高斯消去法是最有影响力的数学家之一的Carl Friedrick Gauss（1777－1855）在研究数据拟合中的最小二乘法的精度问题时开发出来的。参照文献[67]简要了解高斯最小二乘法和消去法。

$$L=\begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ l_{n1} & & \cdots & l_{nn} \end{bmatrix},\quad U=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & & u_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix}$$

三角方程组的求解比对角方程组稍微复杂一点。如下例，上三角方程组的系数矩阵$\boldsymbol{A}$和向量$\boldsymbol{b}$定义为

$$\boldsymbol{A}=\begin{bmatrix} -2 & 1 & 2 \\ 0 & 3 & -2 \\ 0 & 0 & 4 \end{bmatrix},\qquad \boldsymbol{b}=\begin{bmatrix} 9 \\ -1 \\ 8 \end{bmatrix}$$

380

等价于

$$\begin{aligned} -2x_1+x_2+2x_3&=9 \\ 3x_2+-2x_3&=-1 \\ 4x_3&=8 \end{aligned}$$

最后一个方程的解为

$$x_3=\frac{8}{4}=2$$

把已求出的x_3的值代入第二个方程后得到

$$x_2=\frac{1}{3}(-1+2x_3)=\frac{3}{3}=1$$

最后，把x_2和x_3的值代入第一个方程得到

$$x_1=\frac{1}{-2}(9-x_2-2x_3)=\frac{4}{-2}=-2$$

这一过程是把已知量从最后一个方程开始代入，最终把所有已知量代入第一个方程，这一求解过程称为向后代入法，这是求解上三角方程组最简明高效的方法。给定$\boldsymbol{Ux}=\boldsymbol{b}$，其中$\boldsymbol{U}$是$n\times n$上三角矩阵，$\boldsymbol{b}$是$n\times 1$（列）向量，由下列算法可以求出$x$:

算法8.2 向后代入法

```
given U, b
x_n = b_n/u_nn
for i = n-1...1
    s = b_i
    for j = i+1...n
        s = s - u_ij x_j
    end
    x_i = s/u_ii
end
```

一般用向前代入法求解下三角方程组。给定$\boldsymbol{Lx}=\boldsymbol{b}$，其中$\boldsymbol{L}$是$n\times n$下三角矩阵，$\boldsymbol{b}$是$n\times 1$向量，用下列算法可以求出$\boldsymbol{x}$的值: 381

算法8.3　向前代入法

```
given L, b
x_1 = b_1/l_11
for i = 2...n
    s = b_i
        for j = 1...i-1
            s = s - l_{i,j} x_j
        end
    x_i = s/l_{i,i}
end
```

在MATLAB中不需要专门编写函数来求解三角方程组。使用内置运算符 \ 后(参见 8.2.5节)，系统能够自动检测并求解三角方程组，而且省略了消去的步骤。

8.2.3　不选主元的高斯消去法

如前面的例题所示，求解三角方程组比较容易，而且，高斯消去法的目的就是把方程组的系数方阵转换成等价的三角系数矩阵。转换成三角矩阵后，就能够用向后代入法求出方程组的解

尽管高斯消去法的原理易于理解而且容易编程，但是，要设计出健壮性好且对所有非奇异矩阵都能使用高斯消去法的程序却是一件复杂的任务，为了使消去程序能够高效地运行，也会遇到具有挑战性的问题。对于MATLAB用户来说，反斜杠运算符使用了可靠的算法和高度优化的代码来求解线性方程组。下几节的内容以更复杂的形式渐进地介绍这一算法，目的是让读者更深入地了解反斜杠运算符的计算过程并能够熟练、正确、自信地应用它。

手工计算　手工计算时，通过合并方程组中的方程并分离出未知量来求线性方程组的解。如下列方程组：

$$x_1 + 3x_2 = 5$$
$$2x_1 + 4x_2 = 6$$

382　用第二个方程减去第一个方程的2倍得到：

$$x_1 + 3x_2 = 5$$
$$-2x_2 = -4$$

得到的系数矩阵是一个上三角矩阵。注意到，第二个方程的x_1、x_2系数和等号右边的值都发生了变化。现在，通过求出x_2的值并代入第一个方程求出x_1，可以求出方程组的解。

对于3 × 3方程组的求解，设系数矩阵和等号右边的向量被定义为

$$\boldsymbol{A} = \begin{bmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} -1 \\ -7 \\ -6 \end{bmatrix}$$

我们不考虑加减符号或x_1、x_2,...符号的话，执行消去操作实际上是对矩阵$\boldsymbol{A}$和向量$\boldsymbol{b}$中的元素进行操作，把矩阵$\boldsymbol{A}$和向量$\boldsymbol{b}$中的元素组合成一个增广矩阵

$$\hat{A} = [A\ b] = \left[\begin{array}{ccc|c} -3 & 2 & 1 & -1 \\ 6 & -6 & 7 & -7 \\ 3 & -4 & 4 & -6 \end{array}\right]$$

矩阵中的垂直线不是必需的，它用于区分矩阵A中的元素和向量b中的元素。

消去操作的目的是把垂直线左边的矩阵转换成三角矩阵。这样，对增广矩阵的操作实际上是对原系数矩阵A和向量b中的元素同时操作。对增广矩阵的合法操作包括：给增广矩阵中的某行乘以一个标量和把矩阵中的任意两行相加或相减。

把增广矩阵转换成三角矩阵时，每次只能够转换一列。位于第一行的元素可用于消去第二行和第三行的第一个元素。把上面矩阵中第一行乘以2并加到第二行，第一行加到第三行得到矩阵

$$\tilde{A}_{(1)} = \left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & -2 & 3 & -7 \end{array}\right]$$

下标(1)用于区分 $\tilde{A}_{(1)}$ 和 $\tilde{A}$，虽然它们对应的方程组的解相同，但是它们并不相等。第二行的元素可用于消去第三行第二列的元素。把第三行减去第二行得到

383

$$\tilde{A}_{(2)} = \left[\begin{array}{ccc|c} -3 & 2 & -1 & -1 \\ 0 & -2 & 5 & -9 \\ 0 & 0 & -2 & 2 \end{array}\right]$$

现在得到的系数矩阵是三角矩阵，用向后代入法得到方程组的解为

$$x_3 = \frac{2}{-2} = -1, \quad x_2 = \frac{1}{-2}(-9 - 5x_3) = 2, \quad x_1 = \frac{1}{-3}(-1 - 2x_2 + x_3) = 2$$

图示（cartoon version）　为了对任意方阵使用高斯消去法，需要用下标(i、j)来引用增广矩阵中的元素。在逐行消去的时候，看看这一过程的算法（还没有指定）图示。图8-8描述了4×4矩阵求解过程中用高斯消去法求解的过程。为了方便起见，增广矩阵中的元素用x来表示，这些元素的值任意，且不要求它们相等。

$$\begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ x & x & x & x & x \\ x & x & x & x & x \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \\ x & x & x & x & x \end{bmatrix} \rightarrow \begin{bmatrix} \boxed{x} & x & x & x & x \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \\ 0 & x' & x' & x' & x' \end{bmatrix}$$

$$\begin{bmatrix} x & x & x & x & x \\ 0 & \boxed{x'} & x' & x' & x' \\ 0 & 0 & x'' & x'' & x'' \\ 0 & x' & x' & x' & x' \end{bmatrix} \rightarrow \begin{bmatrix} x & x & x & x & x \\ 0 & \boxed{x'} & x' & x' & x' \\ 0 & 0 & x'' & x'' & x'' \\ 0 & 0 & x'' & x'' & x'' \end{bmatrix}$$

$$\begin{bmatrix} x & x & x & x & x \\ 0 & x' & x' & x' & x' \\ 0 & 0 & \boxed{x''} & x'' & x'' \\ 0 & 0 & 0 & x''' & x''' \end{bmatrix}$$

图8-8　4×4方程组的高斯消去法的图示。增广矩阵中的元素用x表示，不要求它们相等。撇号的个数表示这个元素被更改的次数。用方框括起来的元素是消去时所选的主元

384

主元所处的行用于消去它下面那些行的第一个非零元素。主元行 (pivot row) 处于主对角线上的元素称为主元素(pivot element)、或简称为主元(pivot)。在图8-8中的主元都用方框框起来了。用来消去它下面的行所对应元素时，在把与主元处在同一列的元素变成零的同时还改变了此行中其他元素的值，用撇号个数表示它们改变的次数。其中，越靠下的元素，被修改的次数越多。

算法　通过分析，总结前几小节介绍过的消去法特点可以总结出算法。从手工计算我们已经知道，用主元行乘以某个标量后、再从其他行中减去所得行可以消去该行中与主元所在列对应的元素，而且，高斯消去法的计算流程也已经在图8-8中表示出来了。

采用第4章介绍的逐步求精的方法来展开介绍这一算法。逐步求精法首先是用一条语句描述所要执行的任务，然后把这个高级的描述细分成一系列更小的子任务并对子任务进行递归细分，直到把子任务分解成能够直接用语句实现的小任务。在开发高斯消去法的过程中，我们借鉴了逐步求精的变种方法。在每次求精的步骤中，我们使用了一种新的嵌套循环结构，这样的话，计算的每个步骤能够清晰地表示出来。消去法处理过程的高级描述如下所示：

把增广矩阵变换成上三角形式。

由于增广矩阵是n行$n+1$列的矩阵，所以最后得到的是个矩形的矩阵，也即，只能前n列n行构成的矩阵为三角矩阵。

可以对增广矩阵按行操作转换成矩形形式。主元行乘以一个标量后可用来消去其他行与主元对应的非零元素。第一行的主元用来消去第二行到第n行的第一个元素、第二行的主元用来消去第三行到第n行的第二个元素，依此类推。对这些行的操作可以用高级算法语句表示成：

```
for i = 1...n − 1
  use pivot row i to zero elements in column i,  for rows i+1 to n
end
```

385　当用第$n-1$行消去第n行、第$n-1$列的元素时，说明消去操作已经结束，因此，第n行不作为主元行，外循环从$i=1$到$i=n-1$。

语句"for rows i+1 to n"隐含说明了这一操作要用另一个循环来实现。假设k是循环的索引（为什么不是i？），得到如下循环：

```
for i = 1...n − 1
  for k = i+1...n
    use pivot row i to eliminate element ã_ki
  end
end
```

其中$\tilde{a}_{ki}$上面的记号(~)说明是对增广矩阵$\tilde{A}$操作，而不是对系数矩阵A操作。主元下面的同列元素对应的列下标$j=i$，所以i是主元所在行的行下标，也是要消去的元素对应的列下标。

对元素$\tilde{a}_{ki}$的消去操作也必须作用于第k行所有其他元素的操作，这就需要另一个循环。当主元的行下标是i时，说明从第一列到第$i-1$列的所有主元以下元素都是0。最内部的行操作是对第k行第i列到第$n+1$列的元素操作。（为什么不是第i列到第n列？）于是可得到如下程序：

```
for i = 1...n − 1
  for k = i+1...n
    for j = i...n+1
      subtract a multiple of ã_ij from ã_kj
    end
  end
end
```

注意到最内部的循环操作还包括把元素 $\tilde{a}_{ki}$ 消去，计算中可以把 $\tilde{a}_{ki}$ 的值直接置为零，用不着计算一次。而且，真要计算的话，由于舍入误差的影响，$\tilde{a}_{ki}$ 的值可能近似为零而并不等于零。

最后一步就是要推导出"$\tilde{a}_{ij}$ 的倍数"对应的公式，它在第k行减去第i行时使 $\tilde{a}_{ki}$ 为零。换句话说，找出满足下列式子的f值

386

$$\tilde{a}_{ki} - f\,\tilde{a}_{ii} = 0$$

求出f的值为

$$f = \frac{\tilde{a}_{ki}}{\tilde{a}_{ii}}$$

把主元所在的第i行的元素值乘以f 再减第k行，可把 $\tilde{a}_{ki}$ 的值置为零。所以，上面的算法对应下列替换操作

$$\tilde{a}_{kj} \leftarrow \tilde{a}_{kj} - \left(\frac{\tilde{a}_{ki}}{\tilde{a}_{ii}}\right)\tilde{a}_{ij} \tag{8-20}$$

把方程（8-20）代入前面的代码段的最内层循环中，得到最终的消去算法如下：

算法8.4 高斯消去法

```
Input: A, b
form Ã = [A b]
for i = 1...n − 1
  for k = i+1...n
    for j = i...n+1
      ã_kj = ã_kj − (ã_ki / ã_ii) ã_ij
    end
  end
end
```

程序清单8-1中的函数GEshow是用向后代入的高斯消去法的MATLAB实现。下列语句用GEshow函数求出了8.2.3节中的3 × 3方程组：

```
>> A = [-3  2  -1; 6  -6  7; 3  -4  4];    b = [-1; -7; -6];
>> x = GEshow(A,b)
```

上式计算的结果与前面手工计算得到得结果一致。在每次消去的时候，GEshow函数会打印出增广矩阵的值，这对示范消去法的求解思路是很有用的。对于线性方程组，可以用8.2.5节介绍的反斜杠运算符求解。

387

程序清单8-1 函数GEshow用高斯消去法求解方程组并打印出消去过程和向后代入过程的结果，不选主元

```
function x = GEshow(A,b,ptol)
% GEshow  Show steps in Gauss elimination and back substitution
%         No pivoting is used.
%
% Synopsis:  x = GEshow(A,b)
%            x = GEshow(A,b,ptol)
%
% Input:     A,b  = coefficient matrix and right hand side vector
%            ptol = (optional) tolerance for detection of zero pivot
%                   Default:  ptol = 50*eps
%
% Output:    x = solution vector, if solution exists

if nargin<3, ptol = 50*eps;  end
[m,n] = size(A);
if m~=n,  error('A matrix needs to be square');  end
nb = n+1;   Ab = [A b];    %  Augmented system
fprintf('\nBegin forward elmination with Augmented system:\n');  disp(Ab);

% --- Elimination
for i = 1:n-1                   %  loop over pivot rows
  pivot = Ab(i,i);
  if abs(pivot)<ptol, error('zero pivot encountered');  end
  for k = i+1:n                 %  k = index of next row to be eliminated
     Ab(k,i:nb) = Ab(k,i:nb) - (Ab(k,i)/pivot)*Ab(i,i:nb);
  end
  fprintf('\nAfter elimination in column %d with pivot = %f\n',i,pivot);
  disp(Ab);
end

% --- Back substitution
x = zeros(n,1);               %  preallocate memory for and initialize x
x(n) = Ab(n,nb)/Ab(n,n);
for i=n-1:-1:1
  x(i) = (Ab(i,nb) - Ab(i,i+1:n)*x(i+1:n))/Ab(i,i);
end
```

388

8.2.4 选主元的高斯消去法

若增广矩阵为非奇异矩阵，按行操作时，导致对角线元素中出现零，那么就不能用算法8.4求方程组的解（算法失效）。本节给出了一个出现这种失效的例子，并提出了失效出现后的补救措施。

用前面介绍的消去程序求解下列方程组

$$\boldsymbol{A}=\begin{bmatrix}2 & 4 & -2 & -2\\ 1 & 2 & 4 & -3\\ -3 & -3 & 8 & -2\\ -1 & 1 & 6 & -3\end{bmatrix},\quad \boldsymbol{b}=\begin{bmatrix}-4\\ 5\\ 7\\ 7\end{bmatrix}$$

注意到这个方程组没有任何错误。系数矩阵是非奇异矩阵、向量b在A的列空间中，所以方程组有唯一解。

消去之前首先构造一个增广矩阵

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 1 & 2 & 4 & -3 & 5 \\ -3 & -3 & 8 & -2 & 7 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right] \tag{8-21}$$

第一行乘以1/2再减第二行、第一行乘以3/2再加第三行、第一行乘以1/2再加第四行得到

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 0 & 5 & -2 & 7 \\ 0 & 3 & 5 & -5 & 1 \\ 0 & 3 & 5 & -4 & 5 \end{array}\right]$$

到下一步消去的时候将会出现错误，因为$\tilde{a}_{22}=0$，所以第二行不能用于消去第三行和第四行的对应元素$\tilde{a}_{32}$和$\tilde{a}_{42}$，由消去公式$a_{i,j} = a_{i,j} - a_{i,j}/a_{22}$得到零除的结果。这个问题在开始消去的时候并不会明显地反映出来，这就要求算法能够检测出主元是否为零，若为零，该如何处理？通常可以对增广矩阵做行交换操作，然后继续执行消去操作。我们知道行的交换操作不会改变方程组的解。

这种在消去过程中交换行以避免出现主元为零的过程称为局部主元，或简称为选主元(pivoting)。这个术语容易造成混淆，不要把消去过程中定义的与主元行相关的操作与选主元混淆，主元行涉及主元的正常操作不叫做选主元，局部选主元是指为了把我们期望的元素交换到主元的位置。在本例中，交换第二行与第四行可以得到局部选主元[⊖]。结果是

389

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 3 & 5 & -5 & 1 \\ 0 & 0 & 5 & -2 & 7 \end{array}\right]$$

然后，用第二行减第三行得到：

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 0 & 0 & -1 & -4 \\ 0 & 0 & 5 & -2 & 7 \end{array}\right]$$

这时，在主元的位置上又出现了零。那么，再交换第三行与第四行得到：

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 0 & 5 & -2 & 7 \\ 0 & 0 & 0 & -1 & -4 \end{array}\right] \tag{8-22}$$

⊖ 也可以交换第二行与第三行。

此时的系数矩阵是三角矩阵，用向后代入法可以求出方程组的解。

行交换与列交换　上面的例题说明行交换能够避免主元素位置出现零。同理，也可以进行列交换，或者行交换与列交换同时进行。任何交换都可以进行，但是，前提是要保证交换后不改变方程组的解。执行行交换操作相当于改变源方程组中各个方程的顺序；而列交换操作相当于改变解向量中未知量的顺序。

下面是一个进行列交换的例子，系数矩阵是4 × 4的矩阵，假设元素a_{11}用来消去它下面每行对应的元素：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & a'_{22} & a'_{23} & a'_{24} \\ 0 & a'_{32} & a'_{33} & a'_{34} \\ 0 & a'_{42} & a'_{43} & a'_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

假设在消去a_{21}的过程中，发现a'_{22}的值是零或足够小，且a'_{24}可作为主元，那么可以交换第二列与第四列的元素。下图中用方框框起来的元素都是将要被交换的元素，如：

[390]

$$\begin{bmatrix} a_{11} & \boxed{a_{12}} & a_{13} & \boxed{a_{14}} \\ 0 & \boxed{a'_{22}} & a'_{23} & \boxed{a'_{24}} \\ 0 & \boxed{a'_{32}} & a'_{33} & \boxed{a'_{34}} \\ 0 & \boxed{a'_{42}} & a'_{43} & \boxed{a'_{44}} \end{bmatrix} \begin{bmatrix} x_1 \\ \boxed{x_2} \\ x_3 \\ \boxed{x_4} \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

列交换后，得到的方程组是

$$\begin{bmatrix} a_{11} & a_{14} & a_{13} & a_{12} \\ 0 & a'_{24} & a'_{23} & a'_{22} \\ 0 & a'_{34} & a'_{33} & a'_{32} \\ 0 & a'_{44} & a'_{43} & a'_{42} \end{bmatrix} \begin{bmatrix} x_1 \\ x_4 \\ x_3 \\ x_2 \end{bmatrix} \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \\ b'_4 \end{bmatrix}$$

矩阵$\boldsymbol{A}$中的列和$\boldsymbol{x}$中元素都进行了重排操作，但是向量$\boldsymbol{b}$没变。

列交换比行交换更难于用编程实现。许多实际问题中得到的矩阵，它们作行交换后都能够保证正向消去能够顺利进行。当同时进行列交换与行交换时，称为满选主元（full pivoting）或全选主元（complete pivoting）。

上面的例题中，当主元为零时进行交换操作，即选主元操作。具有健壮性的高斯消去算法还使用了另外一种积极的选主元机制，它把处于同一列中元素最大的行交换到主元行（这样可以避免先去计算主元是否为零，然后才能进行交换决策）。例8.6说明了这种机制的优点。对方程（8-21）使用这一机制时，在第一次消去之前，先作行交换。交换第一行与第三行得到：

$$\tilde{\boldsymbol{A}} = \left[\begin{array}{cccc|c} -3 & -3 & 8 & -2 & 7 \\ 1 & 2 & 4 & -3 & 5 \\ 2 & 4 & -2 & -2 & -4 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

用第一行消去其他行对应元素得到

$$
\tilde{\boldsymbol{A}}=\left[\begin{array}{cccc|c}
-3 & -3 & 8 & -2 & 7 \\
0 & 1 & 6\frac{2}{3} & -3\frac{2}{3} & 7\frac{1}{3} \\
0 & 2 & 3\frac{1}{3} & -3\frac{1}{3} & \frac{2}{3} \\
0 & 2 & 3\frac{1}{3} & -2\frac{1}{3} & 4\frac{2}{3}
\end{array}\right]
$$

391

此时，虽然 $\tilde{a}_{22}$ 不为零，但是需要交换第二行与第三行的元素:

$$
\tilde{\boldsymbol{A}}=\left[\begin{array}{cccc|c}
-3 & -3 & 8 & -2 & 7 \\
0 & 2 & 3\frac{1}{3} & -3\frac{1}{3} & \frac{2}{3} \\
0 & 1 & 6\frac{2}{3} & -3\frac{2}{3} & 7\frac{1}{3} \\
0 & 2 & 3\frac{1}{3} & -2\frac{1}{3} & 4\frac{2}{3}
\end{array}\right]
$$

把第二行作为主元行继续执行消去操作得到:

$$
\tilde{\boldsymbol{A}}=\left[\begin{array}{cccc|c}
-3 & -3 & 8 & -2 & 7 \\
0 & 2 & 3\frac{1}{3} & -3\frac{1}{3} & \frac{2}{3} \\
0 & 0 & 5 & -2 & 7 \\
0 & 0 & 0 & 1 & 4
\end{array}\right]
$$

这时，第四行的元素 $\tilde{a}_{43}$ 为零，不需要再消去。尽管得到的三角系数矩阵与方程（8-22）中的不同，但是它们的解是相同的。

例8.6 小值主元对矩阵变换的影响

前面一节说明了主元为零时，消去无法进行。当主元值是个很小的数时，也会出现错误。本例中的计算摘自文献[28]。

求解下列方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 时，增广矩阵定义为

$$
\tilde{\boldsymbol{A}}=\left[\begin{array}{cc|c}
0.0001 & 0.5 & 0.5 \\
0.4 & -0.3 & 0.1
\end{array}\right]
$$

为了说明有限精度对计算的影响，每步消去和向后代入运算时精确到4位有效数字。精确到4位有效数字的精确解是 $\boldsymbol{x}=[0.9999,\ 0.9998]^T$。

由于 $\tilde{a}_{11}\neq 0$，所以第一行乘以0.4/0.0001 = 4000后从第二行中减去，消去 $\tilde{a}_{21}$，乘法过程没出现精度丢失，第二行的非零元素变为:

$$
\tilde{a}_{22}=-0.3-(4000)(0.5)=-2000, \qquad \tilde{a}_{23}=0.1-(4000)(0.5)=-2000
$$

增广矩阵变换成

$$
\tilde{\boldsymbol{A}}=\left[\begin{array}{cc|c}
0.0001 & 0.5 & 0.5 \\
0 & -2000 & -2000
\end{array}\right]
$$

392

使用4位精度的算术运算进行向后代入后得到

$$x_2 = \frac{-2000}{-2000} = 1, \qquad x_1 = \frac{1}{0.0001}(0.5-(0.5)(1)) = 0$$

这个结果与精确解相差了很多。

这是因为计算 $\tilde{a}_{22}$ 和 $\tilde{a}_{23}$ 时引起了很大的误差。主要原因是主元太小，所以要消去第二行的第一个元素所需乘数特别大，减法操作导致第二行元素的精度严重丢失（丢失了所有有效位）。在消去过程中出现严重的精度丢失造成的结果是它与下面这个与原增广矩阵不同的增广矩阵有相同的精确解（$x_1=0, x_2=1$）：

$$\tilde{A}' = \left[\begin{array}{cc|c} 0.0001 & 0.5 & 0.5 \\ 0.4 & 0 & 0 \end{array}\right]$$

除了矩阵 $\tilde{A}$ 在消去过程中由于很小的扰动（$\sim 5\times 10^{-5}$）造成产生了相同的解以外，矩阵 $\tilde{A}$ 和 $\tilde{A}'$ 所代表的方程组完全不同。当然，通过行交换可以修正小值主元产生的错误。

行交换后得到的增广矩阵是

$$\tilde{A} = \left[\begin{array}{cc|c} 0.4 & -0.3 & 0.1 \\ 0.0001 & 0.5 & 0.5 \end{array}\right]$$

用第一行乘以$0.0001/0.4 = 2.500\times 10^{-4}$减第二行可以消去 $\tilde{a}_{21}$。这时，第二行的非零元素成为：

$$\tilde{a}_{22} = 0.5-(2.500\times 10^{-4})(-0.3) = 0.5-7.5\times 10^{-5} = 0.5000$$
$$\tilde{a}_{23} = 0.5-(2.500\times 10^{-4})(-0.1) = 0.5-7.5\times 10^{-5} = 0.5000$$

结果：

$$\tilde{A} = \left[\begin{array}{cc|c} 0.4 & -0.3 & 0.1 \\ 0.0 & 0.5 & 0.5 \end{array}\right]$$

向后代入得到：

$$x_2 = \frac{0.5}{0.5} = 1$$
$$x_1 = \frac{1}{0.4}(0.1+(0.3)(1)) = 1$$

它与精确解很近似。

[393] 这个例子说明在执行消去之前选最大的列元素作为主元能够减小舍入误差。实际上，在用高斯消去法编程时，也是在执行消去前先进行行交换以选取最大元素作为主元，并不是在发现主元为零时才进行交换操作。

在参照文献Watkins [78, 2.6节]中非正规但明晰地介绍了小值主元是如何破坏三角矩阵A中的元素的精度的。

实现　下面的算法实现了上面提到的采用行交换的选主元机制。

算法8.5　局部选主元的高斯消去法

Input: $\boldsymbol{A}, \boldsymbol{b}$
form $\tilde{\boldsymbol{A}} = [\boldsymbol{A}, \boldsymbol{b}]$
for $i=1 \ldots n-1$

find i_p such that $\left|\tilde{a}_{i_p,i}\right| = \max(|\tilde{a}_{k,i}|)$ for k=$i \ldots n$

exchange row i_p with row i

for $k=i+1 \ldots n$

  for $j=i \ldots n+1$

    $\tilde{a}_{k,j} = \tilde{a}_{k,j} - (\tilde{a}_{k,i} / \tilde{a}_{i,i})\tilde{a}$

  end

 end

end

程序清单8-2中的函数GEPivShow实现了算法8.5，它用三句紧凑的MATLAB语句实现了行交换和主元的选取。下面对这几个语句进行一些解释，内置函数max能够选择出优良的主元，它带两个参数:

```
[pivot,p] = max(abs(Ab(i:n,i)));
```

返回绝对值最大的元素（主元）及其在列向量Ab(i:n,i)中的下标。若最大元素在增广矩阵的主对角线上，那么p = 1。语句ip = p+i-1计算出主元的行下标。下列语句实现行交换:

```
Ab([i ip],:) = Ab([ip i],:);
```

选取主元并交换行后，算法执行的其他过程与函数GEshow的执行过程完全一样。 394

程序清单8-2 函数GEPivShow用局部选主元的高斯消去法求方程组的解并打印出消去过程中每一步的结果

```
function x = GEPivShow(A,b,ptol)
% GEPivShow  Show steps in Gauss elimination with partial pivoting and
%            back substitution
%
% Synopsis:  x = GEPivShow(A,b)
%            x = GEPivShow(A,b,ptol)
%
% Input:     A,b  = coefficient matrix and right hand side vector
%            ptol = (optional) tolerance for detection of zero pivot
%                   Default:  ptol = 50*eps
%
% Output:    x = solution vector, if solution exists

if nargin<3, ptol = 50*eps;  end
[m,n] = size(A);
if m~=n,  error('A matrix needs to be square');  end
nb = n+1;   Ab = [A b];    %  Augmented system
fprintf('\nBegin forward elmination with Augmented system:\n');  disp(Ab);

% --- Elimination
for i = 1:n-1                          %  loop over pivot row
  [pivot,p] = max(abs(Ab(i:n,i)));     %  value and index of largest pivot
  ip = p + i - 1;                      %  p is index in subvector i:n
  if ip~=i                             %  ip is true row index of desired pivot
    fprintf('\nSwap rows %d and %d;  new pivot = %g\n',i,ip,Ab(ip,i));
    Ab([i ip],:) = Ab([ip i],:);       %  perform the swap
  end
```

```
    pivot = Ab(i,i);
    if abs(pivot)<ptol, error('zero pivot encountered after row exchange'); end
    for k = i+1:n            %  k = index of next row to be eliminated
       Ab(k,i:nb) = Ab(k,i:nb) - (Ab(k,i)/pivot)*Ab(i,i:nb);
    end
    fprintf('\nAfter elimination in column %d with pivot = %f\n',i,pivot);
    disp(Ab);
  end

  % --- Back substitution
  x = zeros(n,1);           %  preallocate memory for and initialize x
  x(n) = Ab(n,nb)/Ab(n,n);
  for i=n-1:-1:1
    x(i) = (Ab(i,nb) - Ab(i,i+1:n)*x(i+1:n))/Ab(i,i);
  end
```

395

8.2.5 用反斜杠运算符求解方程组

在MATLAB中，通过“左除”可以求出方程组$Ax = b$的解，用反斜杠运算符能够表示左除这一操作（通过左乘A的逆来进行算术操作）。首先，参照标量方程的求解方法：

$$5x = 20 \quad \Rightarrow \quad x = (5)^{-1}20 = 4$$

符号$(5)^{-1}$不方便使用，但是它表示了方程的左右两边都左乘$(5)^{-1}$能够求得方程的解这一事实。扩展到方程组后，得到方程组的解法为：

$$Ax = b \quad \Rightarrow \quad x = A^{-1}b$$

$A^{-1}b$是$Ax = b$方程组的形式化解。用MATLAB符号表示时，可以用系数矩阵A和等号右边的向量b求解

$$x = A\backslash b$$

符号$A\backslash$...表示左乘矩阵A的逆。

\ 运算符不是通过计算A^{-1}来求方程组的解，而是通过测试矩阵A的形状等其他性质来选取有效的求解算法。8.4.3节介绍了如何用 \ 运算符求方程组的解。对于$n \times n$矩阵，\ 运算符选取局部选主元和向后代入策略的高斯消去法的LU分解求方程组的解。而且，\ 运算符实现了算法8.5的高效且健壮性更好的版本。如下例所示：

```
>> A = [1 2 4; 1 3 9; 1 4 16];
>> b = [2 4 7]';
>> x = A\b

x =
    1.0000
   -0.5000
    0.5000
```

在MATLAB中，使用 \ 运算符求方程组的解是一种常用的方法（推荐使用！）。

例8.7　求解泵曲线构成的方程组的解

例8.1中根据泵曲线模型的二次内插式得到一个3 × 3的方程组。本例介绍如何使用 \ 运算符求解这个方程组。给定三组（q，h）数据对，方程组如下所示：

396

$$\begin{bmatrix} q_1^2 & q_1 & 1 \\ q_2^2 & q_2 & 1 \\ q_3^2 & q_3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \tag{8-23}$$

系数矩阵是范特蒙德矩阵（Vandermonde matrix），它是用单项式基本插值式来构造插值时产生的（参照10.2.1节）。

程序清单8-3中的函数pumpCurve用于建立方程组（8-23）并求出它的解。用例8.1中的数据测试函数pumpCurve的正确性：

```
>> q = [1; 8; 14]*1e-4;
>> h = [115; 110;  92.5];
>> c = pumpCurve(h,q);
>> fprintf('  %14.6e\n',c)
   -1.694139e+07
    8.104396e+03
    1.143590e+02
```

因为向量c中元素值的变化范围超出了10^5，所以用科学记数法表示向量c，这是由插值公式决定的。例如，用单项式代替牛顿多项式进行插值（参照10.2.3节）。习题32~34将讨论使用单项式插值、公式会产生什么变化。

程序清单8-3 用函数pumpCurve求解二次泵曲线的方程组

```
function c = pumpCurve(h,q)
% pumpCurve  Coefficients of quadratic pump curve given head and flow rate
%
% Synopsis:   c = pumpCurve(h,q)
%
% Input:      h = 3 element row or column vector of head values
%             q = 3 element row or column vector of flow rate values
%
% Output:     c = column vector of coefficients such that
%                  h = c(1)*q.^2 + c(2)*q + c(3)

A = [q(1)^2  q(1)   1;          %  Compute A
     q(2)^2  q(2)   1;
     q(3)^2  q(3)   1];
b = [h(1); h(2); h(3)];         %  Assign values to b
c = A\b;                        %  Solve the system
```

397

8.3 数值法求解方程组$Ax = b$的局限性

8.1.2节介绍了方程组$Ax = b$有解的数学条件。本节再介绍用数值方法求解$Ax = b$的附加限制。其中，有方程组大小的限制，这是由计算机时钟速率和内存的限制造成的。虽然在实际使用中大部分方程组的求解不用考虑硬件的限制，但是需要了解这些限制始终是存在的。

用数值方法求解方程组$Ax = b$的另一个重要的局限性是有限精度计算的限制。在严格的算术意义上，只要矩阵A非奇异，方程组就有解。在有限精度算术中，矩阵A的奇异性始终是存在的，只是程度上的问题。对奇点（singularity）的接近程度用矩阵的条件数来度量。病态矩阵的条件数比较大。对于方程组$Ax = b$，当矩阵A的条件数越大时，方程组的解越不可靠（相

对于A的条件数小而言)。8.3.2节将介绍条件数，8.3.5节介绍在有限精度算术中，求解方程组$Ax=b$的实用方法。

8.3.1 计算量

随着个人电脑的普及和大容量存储器的使用，计算机可以操作的矩阵越来越大。计算机越来越快地向功能更强大而价格更低廉的方向发展。现在，用一台标准配置、中等价格的计算机可以很快地求解包含数百个未知量的方程组。但是，只要时钟速率和内存有限，那么能够求解方程组的大小也有限。求解方程组要一定的时间，存储系数矩阵也需要一定的内存容量，这些就是主要的限制因素。

当矩阵的行数（或列数）增加时，矩阵操作需要的计算量（computer effort）急剧增加。表7-2总结了矩阵和向量的乘法操作所需的浮点操作次数。表8-1总结了求解线性方程组所需要的浮点操作的数量级。操作次数可用来估算计算机求解给定尺寸方程组所需的计算时间。当n很小，如$n<50$时，这些估算都不准确，这是因为在估算计算量时只考虑了影响计算量的最重要项。

显然，运行速度快的计算机比速度慢的计算机求解给定问题的速度快。表8-1中的计算量没有考虑加、减操作以及装载和存储数据的时间。优化数据在内存子系统中的移动能够极大地减少矩阵的运行时间（参见文献[15，2.6节]中的例子）。

398

表8-1 求解线性方程组所需的操作次数。运算量是按运算的数量级来估算的，当N很大时，这种估算有效。在Golub和Van Loan的文献[32]中有关于算法和工作量估算的更详细的信息

算 法	计算量
求解三角方程组（向后代入法或向前代入法）	n^2次浮点操作
高斯消去法	$2n^3/3$次浮点操作
局部选主元的高斯消去法	$2n^3/3$次浮点操作和n^2比较
全选主元的高斯消去法	$2n^3/3$次浮点操作和n^3比较
局部选主元的LU分解	$2n^3/3$次浮点操作和n^2比较
Cholesky分解	$n^3/3$次浮点操作

8.3.2 对输入参数的敏感性

在发明数字计算机时，人们就认为在电脑上用高斯消去法求解方程组所得到的解并不可信。当时，数学家们担心消去和向后代入过程的累计舍入误差会严重影响最后的结果。20世纪40年代晚期，在Goldstine和von Neumann、Turing和Wilkinson的研究论文中就提出：舍入误差是有界限的㊀。现在，数学上的误差界限和良好的计算结果之间还有很大的差距，前者使人们感到悲观，但另一方面，用新实现的高斯消去法往往能够获得良好的计算结果㊁，这又使人们感到鼓舞。

求解方程组中的数值问题归根结底可以分成两个单独的且相关的领域：对输入数据的敏感性和算法的稳定性。对输入的敏感性是指系数矩阵接近奇异矩阵的程度，计算的稳定性是指计算过程中舍入误差的快速累积是否会影响最终结果的可信度。这些问题在下一节将进行详细介绍。

㊀ Goldstine在文献[30]的第4章第3部分简要地介绍了早期的发展。

㊁ 参照Trefethen和Bau在文献[76，第22章]中对此不一致所作的通俗解释。

矩阵A和向量b发生扰动后的影响 假设在消去和向后代入过程中没有舍入误差。即，在存储矩阵A和向量b的元素时不会产生舍入误差，而且在严格的算术意义上求解方程组$Ax = b$。 399

设对于2 × 2方程组，向量$b = [1, 2/3]^T$，由于2/3无法用精确的二进制浮点数表示，所以

```
>> b = [ 1; 2/3 ]
b =
    1.0000
    0.6667
```

在b_2处引入了舍入误差。读者可以看出，下列两个方程组的精确解不同。

$$Ax = \begin{bmatrix} 1 \\ 2/3 \end{bmatrix} \quad 和 \quad Ax = \begin{bmatrix} 1 \\ 0.6667 \end{bmatrix}$$

若A是良态的，那么这两个方程组的解的差别可忽略不计；若A是病态的，那么这两个方程组的解有很大的差别。通过测试A和b发生扰动后方程组$Ax = b$解的变化，读者能够更准确地理解良态与病态的概念。首先考虑b发生扰动的情况。

假设A非奇异，δb中的元素比b中的元素小，$x+\delta x_b$是受扰动后的方程组的精确解，则有：

$$A\ (x+\delta x_b) = b+\delta b \tag{8-24}$$

δx_b中的下标b暗示了是由于b的改变才导致了x也发生变化。由于x是方程$Ax = b$的解，对上面方程左边减去Ax，右边减去b得到$A\,\delta x_b = \delta b$，或者：

$$\delta x_b = A^{-1}\delta b \tag{8-25}$$

直观上看，如果δb很小，或可更精确地表示成：

$$\frac{\|\delta b\|}{\|b\|} << 1$$

那么读者会希望

$$\frac{\|\delta x_b\|}{\|x\|} << 1$$

但是，马上我们可以看到，无法保证$\|\delta x_b\| / \|x\|$足够小。 400

对方程（8-25）取合适的范数[⊖]得到$\|\delta x_b\| = \|A^{-1}\delta b\|$。应用矩阵范数的一致性要求（参照7.2.4节）得到

$$\|\delta x_b\| \leqslant \|A^{-1}\|\|\delta b\| \tag{8-26}$$

类似地，对$Ax = b$两边取合适的范数得到$\|b\| = \|Ax\|$且

$$\|b\| \leqslant \|A\|\,\|x\| \tag{8-27}$$

方程（8-26）和方程（8-27）都是对标量操作，可以用简单的线性代数规则计算。重新排列方程（8-27）得到：

$$\frac{1}{\|x\|} \leqslant \frac{\|A\|}{\|b\|} \tag{8-28}$$

⊖ 任意的范数p都可以

用方程（8-26）乘以方程（8-28）得到：

$$\frac{\|\delta x_b\|}{\|x\|} \leqslant \|A\|\|A^{-1}\|\frac{\|\delta b\|}{\|b\|} \tag{8-29}$$

这个结果很重要，它说明当$\|\delta b\|/\|b\|$很小时，只有在$\|A\|\|A^{-1}\|$是$O(1)$时，才能保证$\|\delta x_b\|/\|x\|$也很小；当矩阵A病态时，$\|A\|\|A^{-1}\|>>1$。

下面对矩阵A发生扰动时进行分析。当A发生很小的扰动得到δA时，$\|\delta A\|/\|A\|$（A和δA形状相同）很小，假设A和$A+\delta A$都非奇异，且$x+\delta x_A$是下列方程组的精确解

$$(A+\delta A)(x+\delta x_A)=b \tag{8-30}$$

上面的方程是线性的，因此

$$Ax+A\,\delta x_A+\delta A\,(x+\delta x_A)=b$$

消去等号左边的Ax和等号右边的b，并把方程组重新排列后得到：

$$\delta x_A=-A^{-1}\delta A\,(x+\delta x_A)$$

对方程取合适的范数并根据一致性要求得到：

$$\begin{aligned}\|\delta x_A\| &= \|A^{-1}\delta A(x+\delta x_A)\|\\ &\leqslant \|A^{-1}\|\|\delta A(x+\delta x_A)\|\\ &\leqslant \|A^{-1}\|\|\delta A\|\|(x+\delta x_A)\|\end{aligned}$$

401

最后一次把方程组重新排列后得到：

$$\frac{\|\delta x_A\|}{\|x+\delta x_A\|} \leqslant \|A\|\|A^{-1}\|\frac{\|\delta A\|}{\|A\|} \tag{8-31}$$

对于给定的x值，δx_A可在零和无穷大之间变化，则得到：

$$0 \leqslant \frac{\|\delta x_A\|}{\|x+\delta x_A\|} \leqslant 1$$

方程（8-31）左部分表明只有$\|A\|\|A^{-1}\|$是O（1）时，δA产生的扰动才会引起x很小的变化。

当A和b同时发生扰动时，分析方法与上面的分析过程类似，分析后得到如下结果（推导过程可参照文献Datta [12]）。

$$\frac{\|\delta x\|}{\|x+\delta x\|} \leqslant \frac{\|A\|\|A^{-1}\|}{1-\|A\|\|A^{-1}\|\dfrac{\|\delta A\|}{\|A\|}}\left[\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right] \tag{8-32}$$

同样，$\|A\|\|A^{-1}\|$因子决定了A和b的扰动对方程组解的影响。

线性方程组的条件数　上一节提到了条件数，它的定义为

$$\kappa(A)\equiv\|A\|\|A^{-1}\| \tag{8-33}$$

它说明当A和b扰动时，方程组的解的变化情况（敏感性）。可以用p范数来衡量条件数的大小。一般用2范数，但是，估算条件数大小的有效程序是基于1范数的。任何范数的范围都是

$$1 \leqslant \kappa(A) \leqslant \infty$$

不考虑p范数的情况，当矩阵是单位矩阵时，范数取下限值（参照习题30）。当矩阵A奇异时，$\kappa(A)=\infty$。当$\kappa(A)\to\infty$时，A逐渐成为病态矩阵。

在精确的算术运算中，能够明确地区分矩阵的奇异性。当A非奇异时，$Ax=b$有解。对于精确运算，奇异和非奇异有清楚的界限。相反，在浮点运算中，矩阵的奇异性总是存在，只是程度上的问题。条件数用来表示矩阵在数字上的奇异程度，条件数越大，越接近奇异矩阵。当$\kappa(A)=\infty$时，矩阵A是完全的奇异矩阵，即相当于精确算术运算中定义的奇异矩阵。

当$\kappa(A)$增加时，方程（8-29）、（8-31）和（8-32）中列出方程的解受扰动的影响加剧。将A和b的元素存入存储器时，可能会出现扰动。当$\kappa(A)$很大时，系数矩阵中任何元素的很小的改变将导致方程组计算出来的解发生很大的变化。8.3.5节介绍$\kappa(A)$如何才称为"大"。 402

读者需要注意，条件数可从数值计算中导出。它的中心思想是，条件数反映了方程组的解受输入数据扰动的影响程度。另一个关键思想是，条件数和所求解的问题有关，和求解算法无关。若所求问题是病态的，那么不管用什么算法，方程的数值解对于输入数据都是敏感的。

例8.8 用一个简单矩阵来作检验

对例8.4中最后的矩阵

$$A=\begin{bmatrix} 2 & 1 \\ 2+\delta & 1 \end{bmatrix}$$

其中，δ是个很小的参数。当$\delta=0$时，矩阵A是奇异的（为什么？）。容易证明

$$A^{-1}=\frac{1}{\delta}\begin{bmatrix} -1 & 1 \\ 2+\delta & -2 \end{bmatrix}$$

且

$$\|A\|_1 = 4+\delta, \qquad \|A^{-1}\|_1 = \frac{3}{\delta}+1$$

结果

$$\kappa_1(A) = \frac{12}{\delta}+7+\delta$$

因此，当$\delta\to 0$时，$\kappa_1(A)\to\infty$。

例8.9 一个良态方程组和一个病态方程组

本例说明了当等号右边的向量发生扰动时，良态方程组和病态方程组的解是如何变化的。用内置命令cond可以求出矩阵的条件数，条件数的计算将在8.3.6节中进行介绍。

首先，当系数矩阵是2×2的良态矩阵时

```
>> A = [1 1; 2 3]
A =
     1     1
     2     3

>> cond(A)
ans =
   14.9330
```

403

当$\boldsymbol{b}$ = [1, 2]'时，方程组$\boldsymbol{Ax} = \boldsymbol{b}$的解是$\boldsymbol{x}$ = [1, 0]'。由于$\boldsymbol{b}$ = [1, 2]'等于矩阵$\boldsymbol{A}$的第一列，所以，方程组的解可表示为：

$$\begin{bmatrix}1\\2\end{bmatrix}=1\begin{bmatrix}1\\2\end{bmatrix}+0\begin{bmatrix}1\\3\end{bmatrix}$$

用MATLAB的\运算符也能够求出方程组的解，求解过程如下所示：

```
>> b = [1, 2]
>> x = A\b
x =
     1
     0
```

当这个良态方程组等号右边的向量$\boldsymbol{b}$发生很小的扰动时，方程组的解也产生很小的扰动。

```
>> bp = [1.001; 2];
>> xp = A\bp
xp =
    1.0030
   -0.0020
```

当$\boldsymbol{b}$发生扰动后，所得向量并不与矩阵$\boldsymbol{A}$的第一列元素的值精确匹配，所以方程的解向量中x_2元素的值必定非零。b_1发生0.1%的变化将导致x_1发生0.3%的变化，且x_2的变化的绝对值与x_1变化的绝对值相当。

对于方程组$\boldsymbol{Cx} = \boldsymbol{b}$，当$\boldsymbol{C}$是2×2的病态矩阵时

```
>> C = [1 1; 2 2+100*eps]
C =
     1     1
     2     2

>> cond(C)
ans =
   4.5422e+14
```

向量$\boldsymbol{b}$仍等于矩阵$\boldsymbol{C}$的第一列，因此，方程组的解仍然是$\boldsymbol{x}$ = [1, 0]^T。尽管$\kappa(C)$很大，用MATLAB求出方程组的精确解仍然是：

404

```
>> b = [1; 2]
>> x = C\b
x =
     1
     0
>> x - [1; 0]
ans =
     0
     0
```

当$\boldsymbol{C}$病态时，尽管$\boldsymbol{b}$发生很小的扰动，方程组的解却发生了很大的变化。当$\boldsymbol{b}'$=[1+δ, 2]'，δ = 0.01时，有：

```
>> bp = [1.001;  2];
>> xp = C\bp
xp =
   1.0e+10 *
    9.0072
   -9.0072
```

注意到，扰动方程组解的两个元素之和约为1：

```
>> xp(1)+xp(2)
ans =
    1.0010
```

下列线性组合要求x_1和x_2的值都很大且可以互相消去：

$$\begin{bmatrix}1.001\\2\end{bmatrix}=x_1\begin{bmatrix}1\\2\end{bmatrix}+x_2\begin{bmatrix}1\\2+100\varepsilon_m\end{bmatrix}$$

8.3.3 计算稳定性

前面一节中提到了病态方程组如何受输入扰动的影响。由实验测得的$\boldsymbol{A}$和$\boldsymbol{b}$的元素由于测量误差可能会产生扰动，系数矩阵计算和存储的时候由于舍入误差也会引起扰动。对扰动敏感是由问题本身的特性决定的，和算法的选择没有关系。这可重新表述为：方程组的状态与使用的算法是独立的。

我们已经知道了限制数值方程解精度的原因，现在是该了解求解算法的时候了。在实际应用中，求解方程组$\boldsymbol{Ax}=\boldsymbol{b}$时，会出现有的算法比其他算法计算效果更好的情况吗？要严格地回答这个问题，涉及到具体的方程求解算法对各种引入误差（输入数据误差、消去和代入阶段引入的舍入误差）的放大情况的度量问题。算法在一定程度上会扩大舍入误差，好的算法扩大得比较少（指它的数量级比由于输入数据的不确定性造成的误差数量级小），得到的计算结果在可接受的范围内；差的算法扩大得比较大，以致于计算结果都无法接受。这种不扩大舍入误差的性质称为稳定性。

405

用局部主元高斯消去法并用向后代入策略求解方程组$\boldsymbol{Ax}=\boldsymbol{b}$是稳定的。算法能够求出下列方程组的精确解：

$$(\boldsymbol{A}+\boldsymbol{E})\hat{x}=\boldsymbol{b},\quad 其中\quad \|\boldsymbol{E}\|_\infty\leqslant n^3 g\varepsilon_m\|\boldsymbol{A}\|_\infty \tag{8-34}$$

n是矩阵$\boldsymbol{A}$的行数（或列数），g是增长因子，ε_m是机器精度（参照文献[32]的证明）。增长因子用于衡量矩阵转换成三角矩阵后矩阵元素在数量上的增加情况。方程（8-34）是误差的理论界限（上下限），实际问题中的矩阵的误差界限并不遵守这一限制$^{\ominus}$，理论界限有些太过于悲观了。方程（8-34）隐含地指出了可能会出现很大的误差，但是在实际问题中的误差比这一理论误差小很多。对于使用高斯消去法和向后代入法的程序，可以忽略因子g和n^3，得到实用的误差界限是

$$\boxed{(\boldsymbol{A}+\boldsymbol{E})\hat{x}=\boldsymbol{b},\quad 其中\quad \|\boldsymbol{E}\|_\infty\leqslant \varepsilon_m\|\boldsymbol{A}\|_\infty} \tag{8-35}$$

它有何含义？

方程（8-35）的$(\boldsymbol{A}+\boldsymbol{E})\hat{x}$部分说明用局部选主元的高斯消去法和向后代入法并不能得到原始问题的精确解，但方程（8-35）的$\|\boldsymbol{E}\|_\infty\leqslant\varepsilon_m\|\boldsymbol{A}\|_\infty$部分说明数值计算得到的解与精确解相差不大。根据文献[76，第104页]，局部选主元的高斯消去法能够得到所求问题（问题的描述是正确的）的近乎精确的解。能够得到所求问题（所求问题接近于原始问题）的精确解的算法称为逆向稳定的(backward stable)。要注意的是，这仅是关于数值求解的精确性的一种间接描述。下一节将介绍精度的更进一步的量化问题。

$\ominus$ 若它表示了实际问题的精度，那么高斯消去法与向后代入法就没有意义了。

对所有的系数矩阵来说，不选主元的高斯消去法不是逆向稳定的。但是，对称矩阵和正定矩阵可以用不含选主元功能的高斯消去法求得精确解。

稳定的算法能够保证对良态方程组求出相对精确的解。由方程（8-29）、（8-31）和（8-32）可知，稳定的算法无法保证能得到病态方程组的精确解。在这些例子中，为了便于分析输入扰动造成的影响，没有考虑舍入误差问题。

406

8.3.4 残差

假设 $\hat{x}$ 是方程组 $Ax = b$ 的数值解，由于问题的病态特性、高斯消去和向后代入时的舍入误差，导致 $\hat{x}$ 与 x 不同，残差向量为：

$$r=b-A\hat{x} \tag{8-36}$$

它用于衡量所求解结果满足原始方程组的程度。根据 r 的定义，若 $\hat{x}$ 接近精确解，那么 $\|r\|$ 接近零，反过来，当 $\|r\|$ 接近零，却不能保证 $\hat{x}$ 是精确解。通过下面的式子不难说明这一点（参见习题36）

$$\frac{\|\hat{x}-x\|}{\|\hat{x}\|} \leqslant \kappa(A)\frac{\|r\|}{\|b\|} \tag{8-37}$$

$\kappa(A)$ 是方程（8-33）定义的矩阵 A 的条件数。同样，方程（8-37）中的范数是任意的 p 范数。

方程（8-37）的左边是解的相对误差。右边说明了在鉴定 r 的大小时哪些组成因素要多加关注。b 为已知量，$\|b\|$ 容易计算出来。暂时忽略方程（8-37）中的 $\kappa(A)$ 因子，当 $\|r\|/\|b\| \ll 1$ 时，说明 $\hat{x}$ 与精确解 x 相差不大。$\kappa(A)$ 的存在说明 $\|r\|$ 很小不能保证 $\|\hat{x}-x\|$ 很小。当残差向量很小时能得到一个成功的解的条件是：当且仅当系数矩阵是良态的。当 $\kappa(A)$ 很大时，则无法保证用高斯消去法和向后代入法求出的解 $\hat{x}$ 是接近方程组 $Ax = b$ 的精确解的。

8.3.5 经验法则

通过8.3.2~8.3.4节的介绍得出下列经验法则（rules of thumb）：

- 用局部选主元高斯消去法和向后代入法求方程组 $Ax=b$ 得到一个数值解 $\hat{x}$，即使当 $\kappa(A)$ 很大时，残差向量 $r=b-A\hat{x}$ 仍可能很小。
- 若 A 和 b 的机器精度是 ε_m，用任意选主元的高斯消去法求解方程组 $Ax=b$ 得到的数值解都精确到 d 位，其中：

407

$$d=|\log_{10}(\varepsilon_m)|-\log_{10}(\kappa(A)) \tag{8-38}$$

方程（8-35）可以得到很小的残差，重新排列方程（8-35）并取范数后得到 $\|A\hat{x}-b\|=\|E\hat{x}\|$。因此

$$\|r\| \leqslant \|E\|\|\hat{x}\| \leqslant \varepsilon_m\|A\|\|\hat{x}\|，\text{或者} \frac{\|r\|}{\|A\|\|\hat{x}\|} \leqslant \varepsilon_m$$

由于结果独立于 $\kappa(A)$，应用局部选主元的高斯消去法和向后代入法得到的解的残差向量非常小，虽然得到的解可能还是不精确。从方程（8-37）可以明显看出 $\kappa(A)$ 对精度的影响。

方程（8-38）由方程（8-32）得来。根据$\kappa(A)$，得到

$$\frac{\|\delta x\|}{\|x+\delta x\|} < \frac{\kappa(A)}{1-\kappa(A)\frac{\|\delta A\|}{\|A\|}}\left[\frac{\|\delta A\|}{\|A\|}+\frac{\|\delta b\|}{\|b\|}\right]$$

若A和b的扰动是由存储系数时产生的舍入造成的（即$\|\delta A\| = \varepsilon_m \|A\|$和$\|\delta b\| = \varepsilon_m \|b\|$），那么有：

$$1-\kappa(A)\frac{\|\delta A\|}{\|A\|} \approx 1$$

和

$$\frac{\|\delta x\|}{\|x+\delta x\|} \approx \kappa(A)\varepsilon_m$$

令$10^{-d} = \kappa(A)\varepsilon_m$，得到方程（8-38）。

方程（8-38）为大家提供了一个认识大条件数的重要性的基础。若要得到方程组的有效解，数据的精度d越大越好。用MATLAB计算时，$\varepsilon_m \sim 10^{-16}$，要得到的解为6位精度的话，要求系数矩阵的条件数小于10^{10}。但是，10^{10}仍然是个很大的数（参照习题35）。

8.3.6 计算$\kappa(A)$

计算条件数需要很大的计算量。根据方程（8-33）计算时，需要计算A^{-1}，从A求解A^{-1}需要$O(n^3)$数量级的浮点操作次数。若A^{-1}已知，那么用1或∞范数可以算出$\kappa(A)$，这可减少最终计算$\|A\|\|A^{-1}\|$的浮点操作次数。通常，使用SVD算法计算2范数条件数$\kappa_2(A)$，这是因为

$$\kappa(A) = \frac{\max(\sigma_i)}{\min(\sigma_i)}$$

408

σ_i是矩阵A的奇异值（参照Gill 等在文献[28，3.6.2节]中的介绍，可以知道这一结果是如何得来的）。计算$n \times n$矩阵的SVD需要$O(n^3)$浮点操作次数，所以求解$\kappa(A)$计算代价也是很高的。因为求解方程组需要$O(n^3)$浮点操作次数，所以当要求计算$\kappa(A)$时，求解方程组$Ax = b$需要的时间加倍。对于小矩阵⊖，$\kappa(A)$的计算代价还可以接受。

一般说来，$\kappa(A)$用于诊断目的，它的精确值并不重要，只要知道其数量级就可以了。常用的两种条件数估算程序是内置函数condest和rcond。rcond是较老一些的程序，用来估算$1/\kappa_1(A)$的值；而condest采用了不同的算法，可直接估算$\kappa_1(A)$。

若A是以一个密集矩阵⊖（dense matrix）存储的，使用 \ 运算符后，系统会自动调用rcond来检测系数矩阵的条件数。若rcond的返回值小于容差，就打印出警告信息。例如：

```
>> A = [1 0; 0 eps]          % a very ill conditioned matrix
A =
    1.0000         0
         0    0.0000

>> A\[1; 1]
```

⊖ $n < 500$，这是2000年时对小矩阵的定义。

⊖ 参照附录B了解有关密集矩阵和稀疏矩阵的存储问题。在MATLAB中，用$A = [\cdots]$创建的矩阵是以密集形式存储的。

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 2.220446e-16.
ans =
   1.0e+15 *
    0.0000
    4.5036
```

注意到函数rcond估算的是$1/\kappa_1(A)$，所以，若返回值特别小，就说明系数矩阵是病态的。对稀疏系数矩阵使用 \ 运算符时，系统不会自动计算矩阵的条件数，此时，需要使用condest函数代替rcond[⊖]。

尽管rcond和condest常用来计算条件数且结果是可信的，但是也可能构造出一些矩阵，使采用这两个程序所估算出的条件数与实际的条件数相差很远。当然，这类反例很少而且在对实际问题求解中一般不会出现（参照文献[36，第4章]对条件数估算的系统介绍）。409

8.4 分解法

在高斯消去法中包括把增广矩阵转换成三角矩阵形式的过程。在消去阶段的工作步骤是把矩阵A分解成等价的下三角矩阵L和上三角矩阵U的乘积，这种计算L、U的过程称为LU分解法或矩阵A的LU分解。若A是对称矩阵且正定，那么可以用比LU分解更高效的Cholesky分解法。本节将分别介绍这两种分解法。运算符 \ 将自动对系数矩阵进行LU分解或Cholesky分解（参见8.4.3节）。

表8-2列出了四种重要的矩阵分解法。QR分解在9.2.3节介绍，SVD分解算法在附录A中做简要介绍。

表8-2 矩阵分解的常用方法。它们都由MATLAB内置函数实现（参见lu、chol、qr和svd）

因式分解	名 字	分解的限制条件和它的性质
$A = LU$	LU	A是方阵，非奇异，L是下三角矩阵，U是上三角矩阵
$A = C^TC$	Cholesky	A是对称矩阵且正定，C是上三角矩阵
$A = QR$	QR	A是$m\times n$矩阵，Q是正交矩阵，R是上三角矩阵
$A = USV^T$	Singular Value Decomposition（SVD）	A是$m\times n$矩阵并可能秩亏，U是$m\times n$正交矩阵，S是$n\times n$对角阵，V是$n\times n$正交矩阵

8.4.1 LU分解

本节分四个小节：引子、公式推导、行交换的实现和lu函数。第一次阅读时第2、3小节可以略读，第1、4小节需要仔细阅读。

410 **引子** 矩阵A的LU分解就是要计算下三角矩阵L和上三角矩阵U：

$$A = LU \tag{8-39}$$

分解本身并不能求出方程组$Ax = b$的解。高斯消去法也仅仅是将增广系数矩阵转换为三角矩阵形式，还需要用向后代入法才能求出方程组的解。类似地用LU分解法需要求解两个三角方程组才能得到$Ax= b$的解。

因为$A = LU$，所以方程组$Ax = b$等价于

⊖ 原则上，稀疏矩阵也可以使用rcond命令，但是不能用 \ 运算符。

$$(LU)\,x = b \qquad (8\text{-}40)$$

把等号左边重新组合得到$L(Ux)=b$。表达式Ux的积是一个列向量（它的行数与x和b相等）。假设$y = Ux$，方程（8-40）变为

$$Ly = b$$

因为L是一个下三角矩阵，用向前代入法可以求出$y = L^{-1}b$（不直接计算L^{-1}）。求出y后，得到的方程组是

$$Ux = y$$

用向后代入法可以求出x。把这些解法综合后得到求解方程组$Ax = b$的算法。

算法8.6　用LU分解法求方程组$Ax = b$的解

把A分解成L和U

求解关于y的方程组$Ly = b$，解出y　　；使用向前代入法

求解关于x的方程组$Ux = y$，解出x　　；使用向后代入法

分解步骤需要运算量的数量级为$O(n^3)$浮点操作次数。前两步（分解A和求解方程组$Ly = b$）等价于高斯消去法对增广系数矩阵的操作。因此，用LU分解法求解方程组需要的计算工作量和高斯消去法相同。但是，当系数矩阵相同，等号右边的向量b不同时，用LU分解和向后代入法求方程组的解更加高效，因为矩阵A的LU分解只需执行一次。与LU分解的计算量相比，向后代入法的计算量可以忽略。

公式推导　矩阵的LU分解算法有多种推导办法。在本节中，采用了一种直观的方法，即通过把矩阵A与消去矩阵相乘来进行分解（参照文献Stewart [68]和Strange [72]）。Gill等人在文献[28]中给出了另一种简明的推导方法。　[411]

思考由下列矩阵A和向量b定义的方程组：

$$A = \begin{bmatrix} -3 & 2 & -1 \\ 6 & -6 & 7 \\ 3 & -4 & 4 \end{bmatrix}, \qquad b = \begin{bmatrix} -1 \\ -7 \\ -6 \end{bmatrix}$$

这个方程组与8.2.3节中介绍高斯消去法用到的方程组相同。读者最好能够把那些计算过程与本例中的计算过程作个比较。

首先，定义两个消去矩阵$M_{(1)}$和$M_{(2)}$，先不要管它们是如何得到的。

$$M_{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \qquad M_{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}$$

计算下列矩阵-向量乘积得到：

$$A_{(1)} = M_{(1)}A = \begin{bmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & -2 & 3 \end{bmatrix}$$

$$A_{(2)} = M_{(2)}A_{(1)} = M_{(2)}M_{(1)}A = \begin{bmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{bmatrix}$$

观察可知$A_{(1)}$是用矩阵A第一行进行消去得到的矩阵。$A_{(2)}$是用矩阵$A_{(1)}$第二行进行消去得到的矩阵，也是高斯消去法得到的最终结果。所以，乘以矩阵$M_{(1)}$和$M_{(2)}$（按上面的顺序！）等价于对矩阵进行高斯消去。表达式$M_{(1)}A = A_{(1)}$等价于把矩阵A第一列主对角线下面的元素消为零。$M_{(2)}A_{(1)} = A_{(2)}$等价于把矩阵A第二列主对角线下面的元素消为零。注意$M_{(2)}$是和$A_{(1)}$相乘而不是和A相乘。要注意乘以$M_{(i)}$的顺序，因为高斯消去过程中的顺序是很重要的。

高斯消去法与$M_{(i)}$之间的关系不是偶然的。回顾一下消去法，消去一行中的元素的公式为（参照方程（8-20））：

$$a_{kj} \leftarrow a_{kj} - m_{ki}a_{ij}, \qquad j = i+1,\ldots,n$$

其中

$$m_{ki} = \frac{a_{ki}}{a_{ii}}, \qquad j = i+1,\ldots,n \tag{8-41}$$

[412]

是第i行（主元行）用来消去第k行的元素a_{ki}所需要的倍数。方程（8-41）定义了矩阵$M_{(i)}$对应于次对角线上元素的相反数，即：

$$M_{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix}, \qquad M_{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32} & 1 \end{bmatrix} \tag{8-42}$$

上面对3×3矩阵的操作可以扩展到任意大小的$n \times n$矩阵。用一系列消去矩阵乘以方阵能够得到三角矩阵[⊖]

$$M_{(n-1)} \ldots M_{(2)}M_{(1)}A = U \tag{8-43}$$

其中，U是上三角矩阵，且$M_{(i)}$是第i行（主元行）的消去矩阵。

$$M_{(i)} = \begin{bmatrix} 1 & 0 & 0 & \cdots & & & & \\ 0 & 1 & 0 & & & & & \\ \vdots & & \ddots & & & & & \\ & & 0 & 1 & 0 & & & \\ & & & -m_{i+1,i} & 1 & & & \\ & & & -m_{i+2,i} & 0 & 1 & & \\ & & & \vdots & & & \ddots & \\ & & & -m_{n,i} & & & 0 & 1 \end{bmatrix}$$

方程（8-43）变换的目的同高斯消去法一样。把方程（8-43）拆开后能明显看出该等式与LU分解的关系，所以再把矩阵A单独放在方程组等号的左边。把方程（8-43）左右两边同时乘以矩阵$M_{(i)}$的逆可得到矩阵A：

$$\begin{aligned} M_{(n-2)} \ldots M_{(2)}M_{(1)}A &= M_{(n-1)}^{-1}U \\ \ldots M_{(2)}M_{(1)}A &= M_{(n-2)}^{-1}M_{(n-1)}^{-1}U \\ &\vdots \\ A &= M_{(1)}^{-1}M_{(2)}^{-1} \ldots M_{(n-2)}^{-1}M_{(n-1)}^{-1}U \end{aligned} \tag{8-44}$$

⊖ 假设A是非奇异矩阵，乘以$M_{(i)}$后可以得到选出的主元。

把最后结果与方程（8-39）作比较，展现如下关系：

$$L = M_{(1)}^{-1} M_{(2)}^{-1} \cdots M_{(n-2)}^{-1} M_{(n-1)}^{-1} \tag{8-45}$$

413

这个方程很有趣，但它是否有用呢？用方程（8-45）求解矩阵L需要求矩阵的逆并执行计算矩阵乘法，计算量太大，这种求L的方法不可行。求逆运算需要$O(n^3)$浮点操作次数，因此$n-1$个求逆需要$O(n^4)$浮点操作次数。而且，矩阵-矩阵乘积需要另外$O(n^4)$浮点操作次数的计算量。

有两个计算常识（computational fact）可以大大简化方程（8-45）的运算量。为了便于理解，假设系数矩阵还是前例所示使用的3 × 3矩阵（结果可以扩展到$n \times n$矩阵）。直接计算方程（8-42）中$M_{(1)}$和$M_{(2)}$的逆得到（参照习题38）：

$$M_{(1)}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & 0 & 1 \end{bmatrix}, \quad M_{(2)}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & m_{32} & 1 \end{bmatrix} \tag{8-46}$$

因此，计算$M_{(1)}^{-1}$和$M_{(2)}^{-1}$不需要浮点操作！直接计算$M_{(1)}^{-1} M_{(2)}^{-1}$的乘积得到：

$$M_{(1)}^{-1} M_{(2)}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & m_{32} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix}$$

由于矩阵中0、1和非零元素M_{ij}的特殊排列，所以计算$M_{(i)}^{-1}$和乘积$M_{(1)}^{-1} M_{(2)}^{-1} \cdots M_{(n-1)}^{-1}$都不需要进行浮点操作。3 × 3系数矩阵对应的$L$矩阵为：

$$L = \begin{bmatrix} 1 & 0 & 0 \\ m_{21} & 1 & 0 \\ m_{31} & m_{32} & 1 \end{bmatrix} \tag{8-47}$$

这个方程（可以扩展到任意大小的$n \times n$矩阵）说明LU分解得到的L矩阵中只有0、1和高斯消去需要的乘数。乘数都是要计算的，所以计算L需要的浮点操作次数不会多于高斯消去过程所需浮点操作次数。LU分解得到的上三角矩阵U与高斯消去法最后得到的矩阵相同（参照方程（8-43）、（8-44）和（8-45））。因此，LU分解等价于高斯消去法，惟一不同的是对L和U的元素进行不同的处理。后面将会看到，LU分解所得矩阵L和U可以覆盖（overwrite）矩阵A，所以不需要额外的存储空间。

414

通过上面的分析得出LU分解的下列结论：

1. 对于A的每一行

   a) 根据方程（8-41），计算乘数m_{kj}的值，其中$k=i \ldots n$

   b) 利用这些乘数来消去元素a_{kj}，其中$k=i \ldots n$

   c) 把乘数m_{kj}保存在矩阵L的下三角对应位置处

2. 完成消去后，矩阵U包含矩阵A变换后的上三角矩阵。

下列的算法包含了上面的步骤。

算法8.7 LU分解

given A

for $i = 1 \ldots n-1$　　　　　(在主元之间循环)
　for $k = i+1 \ldots n$　　　　　(在主元所在列的下部循环)
　　$\ell_{ki} = a_{ki}/a_{ii}$　　　　　(计算乘数并保存在L中)
　　for $j = i+1 \ldots n$　　　　　(在第k行的元素之间循环)
　　　$u_{kj} = a_{kj} - \ell_{ki}a_{ij}$
　　end
　end
end

算法8.7要求存储和管理矩阵L、U以及A。若LU分解在原位进行(即计算L和U时，把它们保存在矩阵A对应的元素处)，可以简化操作。对于4×4方程组，分解并覆盖矩阵A的过程是:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \to \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \ell_{21} & u_{22} & u_{23} & u_{24} \\ \ell_{31} & a'_{32} & a'_{33} & a'_{34} \\ \ell_{41} & a'_{42} & a'_{43} & a'_{44} \end{bmatrix}$$

$$\to \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \ell_{21} & u_{22} & u_{23} & u_{24} \\ \ell_{31} & \ell_{32} & u_{33} & u_{34} \\ \ell_{41} & \ell_{42} & a''_{43} & a''_{44} \end{bmatrix} \to \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ \ell_{21} & u_{22} & u_{23} & u_{24} \\ \ell_{31} & \ell_{32} & u_{33} & u_{34} \\ \ell_{41} & \ell_{42} & \ell_{43} & u_{44} \end{bmatrix}$$

这里的带单引号的元素，如a'_{32}，表示矩阵A中改变了的元素，即矩阵A中的元素在消去的时候发生了变化。因为矩阵L对角线上的元素都是1，所以不需要存储。修改算法8.7使得L和U的计算在原位进行:

415

算法8.8　在原位进行LU分解

given A
for $i = 1 \ldots n-1$　　　　　(在主元行之间循环)
　for $k = i+1 \ldots n$　　　　　(在主元所在列的下部循环)
　　$a_{ki} = a_{ki}/a_{ii}$　　　　　(计算乘数并保存在A的下半部分)
　　for $j = i+1 \ldots n$　　　　　(在第k行的元素之间循环)
　　　$a_{kj} = a_{kj} - a_{ki}a_{ij}$
　　end
　end
end

程序清单8-4中的函数luNopiv实现了矩阵在原位的LU分解，从函数的名字luNopiv可以知道，这一函数不进行局部选主元所需要的行交换操作。函数的最后两行得到矩阵A原位分解的矩阵L和矩阵U。读者若用luNopiv函数进行原位分解，就不必保留L和U的副本(参照习题40)。下面介绍选主元操作时需要对函数所做的修改。

行交换的实现　如8.2.4节所介绍的，选主元是为了避免在消去阶段出现除数为零(或者一个非常小的数)的情况。LU分解中选主元的思想同高斯消去法中对增广矩阵选主元的思想

程序清单8-4 函数luNopiv分解非奇异方阵成一个单位下三角矩阵 L和一个上三角矩阵U，没有使用选取主元操作

```
function [L,U] = luNopiv(A,ptol)
% luNopiv  LU factorization without pivoting
%
% Synopsis:  [L,U] = luNopiv(A)
%            [L,U] = luNopiv(A,ptol)
%
% Input:     A    = coefficient matrix
%            ptol = (optional) tolerance for detection of zero pivot
%                   Default:  ptol = 50*eps
%
% Output:    L,U = lower triangular matrix, L, and upper triangular
%                  matrix, U, such that A = L*U

if nargin<3, ptol = 50*eps;  end     %  Default tolerance for zero pivot
[m,n] = size(A);
if m~=n,  error('A matrix needs to be square');  end

for i = 1:n-1                         %  loop over pivot rows
  pivot = A(i,i);
  if abs(pivot)<ptol, error('zero pivot encountered');  end
  for k = i+1:n                                   % row k is eliminated next
     A(k,i) = A(k,i)/pivot;                       % compute and store multiplier
     A(k,i+1:n) = A(k,i+1:n) - A(k,i)*A(i,i+1:n); % row ops to eliminate A(k,i)
  end
end

L = eye(size(A)) + tril(A,-1);  %  extract L and U
U = triu(A);
```

一样。它们之间的一个重要的不同点就是：在LU分解阶段中，只能使用系数矩阵，需要记录对系数矩阵所作的行操作，以便在求解过程中对向量$\boldsymbol{b}$做同样的变换。

若用户在消去之前已经知道行交换的排列顺序，那么可以在进行消去处理之前先排列矩阵$\boldsymbol{A}$和向量$\boldsymbol{b}$，它在数学意义上等价于解下面的方程：

$$(\boldsymbol{PA})\,\boldsymbol{x} = \boldsymbol{Pb} \tag{8-48}$$

其中$\boldsymbol{P}$是置换矩阵（参照7.4.8节）。方程（8-48）的解与方程组$\boldsymbol{Ax} = \boldsymbol{b}$的解相同。若在消去的时候进行行交换，那么用LU分解算法得到排列的系数矩阵

$$\boldsymbol{PA} = \boldsymbol{LU} \tag{8-49}$$

其中，$\boldsymbol{P}$、$\boldsymbol{L}$和$\boldsymbol{U}$是输出参数，$\boldsymbol{A}$是输入参数，即

$$\boldsymbol{A} \xrightarrow[\text{LU分解}]{} \boldsymbol{P}、\boldsymbol{L}、\boldsymbol{U}。$$

416

分解以后，需要把矩阵$\boldsymbol{P}$乘以向量$\boldsymbol{b}$，然后才能用因子矩阵$\boldsymbol{L}$和$\boldsymbol{U}$来求方程组关于$\boldsymbol{x}$的解（参照方程（8-48））。

程序清单8-5中的函数luPiv实现了局部选主元的原位LU分解功能。分解完成后创建了因子矩阵$\boldsymbol{L}$和$\boldsymbol{U}$(参照习题40)。采用局部选主元策略的行操作顺序保存在置换向量pv中，而不是

保存在置换矩阵P中。用置换向量pv提供的信息进行行变换等价于用置换矩阵P相乘所进行的行变换，下列表达式是等价的：

$$b \leftarrow Pb \Leftrightarrow \texttt{b = b (pv)}$$

417

表达式b(pv)是数组索引的一个例子（参照3.5.3节）。

程序清单8-5　函数luPiv把非奇异方阵分解成单位下三角矩阵L和上三角矩阵U，使用局部选主元策略

```
function [L,U,pv] = luPiv(A,ptol)
% luPiv  LU factorization with partial pivoting
%
% Synopsis:  [L,U,pv] = luPiv(A)
%            [L,U,pv] = luPiv(A,ptol)
%
% Input:     A    = coefficient matrix
%            ptol = (optional) tolerance for detection of zero pivot
%                   Default:  ptol = 50*eps
%
% Output:    L,U = lower triangular matrix, L, and upper triangular
%                  matrix, U, such that A(pv,:) = L*U
%            pv  = index vector that records row exchanges used to select
%                  good pivots.  The row permutations performed during
%                  elimination can be applied to the right hand side vector
%                  with b(pv).  The L and U returned by luPiv are the
%                  factors of permuted matrix A(pv,:), which is equivalent
%                  to P*A where P is the permutation matrix created
%                  by the two statements P = eye(size(A));  P = P(pv,:).

if nargin<3, ptol = 50*eps;  end     %  Default tolerance for zero pivot
[m,n] = size(A);
if m~=n,  error('A matrix needs to be square');  end
pv = (1:n)';

for i = 1:n-1                          %  loop over pivot row
  [pivot,p] = max(abs(A(i:n,i)));      %  value and index of largest pivot
  ip = p + i - 1;                      %  p is index in subvector i:n
  if ip~=i                             %  ip is true row index of desired pivot
     A([i ip],:) = A([ip i],:);        %  swap the rows
     pv([i ip]) = pv([ip i]);          %  record pivot order
  end
  pivot = A(i,i);
  if abs(pivot)<ptol, error('zero pivot encountered after row exchange');  end
  for k = i+1:n                                    % row k is eliminated next
    A(k,i) = A(k,i)/pivot;                         % compute and store multiplier
    A(k,i+1:n) = A(k,i+1:n) - A(k,i)*A(i,i+1:n);  % row ops to eliminate A(k,i)
  end
end

L = eye(size(A)) + tril(A,-1);  %  extract L and U
U = triu(A);
```

418

下面是函数luPiv的一个应用：

```
>> A = [ 2 4 -2 -2; 1 2 4 -3; -3 -3 8 -2; -1 1 6 -3];
>> b = [-4; 5; 7; 7];
>> [L,U,pv] = luPiv(A)
L =
    1.0000         0         0         0
   -0.6667    1.0000         0         0
   -0.3333    0.5000    1.0000         0
    0.3333    1.0000    0.0000    1.0000
U =
   -3.0000   -3.0000    8.0000   -2.0000
         0    2.0000    3.3333   -3.3333
         0         0    5.0000   -2.0000
         0         0         0    1.0000
pv =
     3
     1
     2
     4
```

给定L、U和pv，用求解三角方程组的算法(算法8.2和算法8.3)和算法8.6实现求解方程组$Ax=b$的第二步和第三步的计算。不需要编写m文件函数，因为内置运算符 \ 能够高效地计算出三角方程组的解。\ 算符包括预处理逻辑，用来判断系数矩阵是否为三角矩阵，若是（或是一个三角矩阵的置换），则不用进行消去和分解，可以直接使用适当的三角方程组的求解算法。因此，由前面的句子给定L、U和pv：

```
>> y = L\b(pv)
y =
    7.0000
    0.6667
    7.0000
    4.0000

>> x = U\y
x =
    1.0000
    2.0000
    3.0000
    4.0000
```

得到了方程组的解。

419

lu函数 内置函数lu采用了局部选主元策略来实现矩阵的LU分解。前面介绍的函数luNopiv和luPiv都是为了说明如何实现LU分解。lu函数采用的是luPiv函数中使用的基本算法的更有效实现，在应用的时候，通常使用lu取代luNopiv和luPiv函数。

函数lu使用局部选主元策略来得到方程（8-49）的分解。可以对置换矩阵P进行显式或隐式处理，这由lu函数的调用形式决定。它有两种调用形式，分别是：

$[L^{(p)},U]$ = lu(A)
$[L,U,P]$ = lu(A)

在第一种形式中，$L^{(p)}$是矩阵A分解所得到的下三角矩阵的置换矩阵形式。根据方程（8-49），得到：

$$A = P^{-1}LU = P^{T}LU = L^{(p)}U$$

$L^{(p)} = P^T L$(为什么$P^{-1} = P^T$?)。由于\运算符能够识别置换三角矩阵，所以$L^{(p)}$有一定的用处。给定表达式$y = L \backslash b$，\运算符检测到L是置换三角矩阵后，析取出其中的置换矩阵作用于向量b并用向前代入法求出y。下面给出如何用算法8.6得出L的隐式置换应用：

```
>> A = [1 2 4; 1 3 9; 1 4 16];
>> b = [2; 4; 7];
>> [L,U] = lu(A)          % Factor A into L and U
L =
    1.0000         0         0
    1.0000    0.5000    1.0000
    1.0000    1.0000         0
U =
     1     2     4
     0     2    12
     0     0    -1

>> y = L\b                % Solve L*y = b
y =
    2.0000
    5.0000
   -0.5000
>> x = U\y                % Solve U*x = y
x =
    1.0000
   -0.5000
    0.5000
```

420

要记住：y =L\b和x=U\y都要依靠\运算符的内部逻辑检测三角矩阵从而使用适当的三角矩阵解法。

lu函数的[L，U，P]=lu (A) 调用形式中，L是严格的下三角矩阵，P是方程（8-49）中出现的（满）置换矩阵。矩阵U在lu函数的两种调用形式中都相同（为什么？）。给定lu函数的L、U和P，用算法8.6的最后两步得到$Ax = b$的解。下面是在MATLAB交互环境中用lu函数和矩阵P计算的例子：

```
>> A = [1 2 4; 1 3 9; 1 4 16];   b = [2; 4; 7];
>> [L,U,P] = lu(A)
>> L
L =
    1.0000         0         0
    1.0000    1.0000         0
    1.0000    0.5000    1.0000

U =
     1     2     4
     0     2    12
     0     0    -1
P =
     1     0     0
     0     0     1
     0     1     0

>> y = L\(P*b)
y =
    2.0000
```

```
     5.0000
    -0.5000

>> x = U\y
x =
     1.0000
    -0.5000
     0.5000
```

如前面所示，没有必要在lu函数的三个参数形式中产生矩阵$\boldsymbol{P}$。而且注意到：给定$\boldsymbol{L}$和$\boldsymbol{U}$，用一行语句可以得到$\boldsymbol{LUx} = \boldsymbol{b}$的解： [421]

```
>> x = U\(L\b)
```

这可以避免在MATLAB的用户工作区创建一个中间向量y，这个中间向量一旦产生，将永久地保存在这个工作区[1]。若$\boldsymbol{L}$和$\boldsymbol{U}$在后面的计算中不需要用到，那么

```
>> x = A\b
```

不用在用户工作区创建矩阵$\boldsymbol{L}$和矩阵$\boldsymbol{U}$也能得到LU分解。

8.4.2 Cholesky分解

若矩阵$\boldsymbol{A}$对称且正定（参照7.4.6节），那么可以得到Cholesky分解：

$$\boldsymbol{A} = \boldsymbol{C}^T\boldsymbol{C}$$

其中，$\boldsymbol{C}$是一个上三角矩阵[2]。此时，假设能实现Cholesky分解的话，则有，$\boldsymbol{Ax} = \boldsymbol{b}$等价于$\boldsymbol{C}^T\boldsymbol{Cx} = \boldsymbol{b}$，求方程组解的过程同LU分解。用Cholesky分解求$\boldsymbol{Ax} = \boldsymbol{b}$的算法如下：

算法8.9 用Cholesky分解法求方程组$\boldsymbol{Ax} = \boldsymbol{b}$的解

分解$\boldsymbol{A}$得到$\boldsymbol{C}$

求关于$\boldsymbol{y}$的方程组$\boldsymbol{C}^T\boldsymbol{y} = \boldsymbol{b}$　　　用向前代入法

求关于$\boldsymbol{x}$的方程组$\boldsymbol{Cx} = \boldsymbol{y}$　　　用向后代入法

Cholesky分解法比LU分解法更加高效，它的计算量是LU分解法的一半；而且对于可靠的高斯消去法，对称正定矩阵不需要采用选主元操作（参照文献[12，定理6.4.2]），所以逻辑操作和数据移动都比LU分解少。

公式推导 Cholesky分解法首先假设方程组有解，然后求解由$\boldsymbol{C}^T\boldsymbol{C} = \boldsymbol{A}$得到的标量方程组。一个4 × 4矩阵的Cholesky分解是： [422]

$$\begin{bmatrix} c_{11} & 0 & 0 & 0 \\ c_{12} & c_{22} & 0 & 0 \\ c_{13} & c_{23} & c_{33} & 0 \\ c_{14} & c_{24} & c_{34} & c_{44} \end{bmatrix} \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} \\ 0 & c_{22} & c_{23} & c_{24} \\ 0 & 0 & c_{33} & c_{34} \\ 0 & 0 & 0 & c_{44} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}$$

注意到，假定矩阵$\boldsymbol{A}$是对称的。用矩阵-矩阵相乘的内积形式得到：乘积可以通过只计算处于矩阵$\boldsymbol{A}$的上三角部分的元素得到，即：

---

[1] 临时向量由MATLAB自动创建和删除。

[2] 很多人将Cholesky分解写成$\boldsymbol{A} = \boldsymbol{GG}^T$，其中$\boldsymbol{G}$是下三角矩阵。

$$
\begin{array}{llll}
i=1, & j=1 & c_{11}^2 = a_{11} & \Rightarrow \quad c_{11} = \sqrt{a_{11}} \\
 & j=2 & c_{11}c_{12} = a_{12} & \Rightarrow \quad c_{12} = a_{12}/c_{11} \\
 & j=3 & c_{11}c_{13} = a_{13} & \Rightarrow \quad c_{13} = a_{13}/c_{11} \\
 & j=4 & c_{11}c_{14} = a_{14} & \Rightarrow \quad c_{14} = a_{14}/c_{11}
\end{array}
$$

$$
\begin{array}{llll}
i=2, & j=2 & c_{11}^2 c_{22}^2 = a_{22} & \Rightarrow \quad c_{22} = \sqrt{a_{22} - c_{12}^2} \\
 & j=3 & c_{11}c_{12} = a_{12} & \Rightarrow \quad c_{23} = (a_{23} - c_{12}c_{13})/c_{22} \\
 & j=4 & c_{11}c_{13} = a_{13} & \Rightarrow \quad c_{24} = (a_{24} - c_{12}c_{14})/c_{22}
\end{array}
$$

$$
\begin{array}{llll}
i=3, & j=3 & c_{13}^2 + c_{23}^2 + c_{33}^2 = a_{33} & \Rightarrow \quad c_{33} = \sqrt{(a_{33} - c_{13}^2 - c_{23}^2)} \\
 & j=4 & c_{13}c_{14} + c_{23}c_{24} + c_{33}c_{34} = a_{34} & \Rightarrow \quad c_{34} = (a_{34} - c_{13}c_{14} - c_{23}c_{24})/c_{33}
\end{array}
$$

$$
\begin{array}{llll}
i=4, & j=4 & c_{14}^2 + c_{24}^2 + c_{34}^2 + c_{44}^2 = a_{44} & \Rightarrow \quad c_{44} = \sqrt{(a_{44} - c_{14}^2 - c_{24}^2 - c_{34}^2)}
\end{array}
$$

根据前面方程的模式我们能得到如下结论:

- C中的元素可以按顺序计算出来
- 对角线上的元素有如下的形式

$$c_{ii} = \sqrt{a_{ij} - (c_{1,i}c_{1,j} + \cdots + c_{i-1,i}c_{i-1,j})} \quad (i=j)$$

- 非对角线上的元素有如下的形式:

$$c_{ij} = (a_{ij} - (c_{1,i}c_{1,j} + \cdots + c_{i-1,i}c_{i-1,j}))/c_{ii}$$

- 对角线和非对角线元素都有下列形式

$$s = a_{i,j} - (c_{1,i}c_{1,j} + \cdots + c_{i-1,i}c_{i-1,j})$$

在计算s时包含了对c_{ij}的求和，它是对矩阵C的部分列的内积，在MATLAB中，这一内积用符号表示为:

```
C(1:i-1,i)'*C(1:i-1,j)。
```

因此，s的计算公式是:

```
s = A(i,j) - C(1:i-1,i)'*C(1:i-1,j)
```

结果

```
C(i,i) = sqrt(s)              % on diagonal terms
```

且

```
C(i,j) = s/C(i,i)             % off-diagonal terms
```

程序清单8-6中的Cholesky函数把这些关系转换成MATLAB代码[⊖]。由于涉及两个小的细节问题的处理使得Cholesky函数的逻辑结构比较复杂。首先，计算$c_{1,1}$时，没有计算内积，这就要求编写一个异常处理部分以避免内积表达式中出现行下标为零的情况:

423

⊖ Cholesky分解的实现是Van Loan在文献[77]中介绍的cholDot函数的变种。

C(1:i-1,i)'*C(1:i-1,j)

其次，能够防止当s是负数时，用sqrt(s)对其进行处理。在数学意义上，当矩阵A不对称且不正定时，s则是负数。在数值意义上，当矩阵A病态或近似正定时，s可能为负数。Cholesky函数中的if s<0测试语句保证了不会出现上面两种情况。

程序清单8-6 对称正定矩阵的Cholesky分解

```
function C = Cholesky(A)
% Cholesky  Cholesky factorization of a symmetric, positive definite matrix
%
% Synopsis:  C = Cholesky(A)
%
% Input:     A = symmetric positive definite matrix
%
% Output:    C = upper triangular matrix such that A = C'*C

[m,n] = size(A);
if m~=n,  error('A must be square');  end
C = zeros(n,n);

for i=1:n
  for j=i:n
    if j==1
      s = A(i,i);   %  i=1, j=1 is special case
    else
      s = A(i,j) - C(1:i-1,i)'*C(1:i-1,j);
    end
    if j>i
      C(i,j) = s/C(i,i);
    else
      if s<=0, error('C is not positive definite to working precision'); end
      C(i,i) = sqrt(s);
    end
  end
end
```

424
425

chol函数 内置函数chol可以得到对称正定矩阵的Cholesky分解。在常规计算的时候，建议读者用内置chol函数而不要用程序清单8-6中的Cholesky函数。下面的一个例子用来示范chol函数的应用。首先，根据经验定义一个对称正定矩阵：

```
>> A = [2 -1 0 0; -1 2 -1 0; 0 -1 2 -1; 0 0 -1 2]
A =
     2    -1     0     0
    -1     2    -1     0
     0    -1     2    -1
     0     0    -1     2
```

然后，得到Cholesky分解：

```
>> C = chol(A)
C =
    1.4142   -0.7071         0         0
         0    1.2247   -0.8165         0
         0         0    1.1547   -0.8660
         0         0         0    1.1180
```

再定义等号右边的向量并用Cholesky分解法求出方程组的解:

```
>> b = [-1 0 0 -2]'
b =
    -1
     0
     0
    -2

>> y = C'\b
y =
   -0.7071
   -0.4082
   -0.2887
   -2.0125

>> x = C\y
x =
   -1.2000
   -1.4000
   -1.6000
   -1.8000
```

使用语句C'*C-A和A\b，我们可以很容易地验证Cholesky分解法用在前面例子的计算中是正确的。

8.4.3 再论反斜杠运算符

反斜杠运算符 \ 是MATLAB内核中复杂问题解决策略的一个简略表示符号。当MATLAB解释器遇到表达式A\b时，它就检查A和b的内容，采取它认为最好的作用过程。下面就是运算符 \ 采用的一些步骤的总结（更多信息见*Using* MATLAB [73]）:

1. MATLAB检查A是否是三角矩阵或其置换矩阵。如果是，就通过合适的代入法来得到解（下三角矩阵用向前代入法，上三角矩阵用向后代入法）。

2. 如果A是方阵，MATLAB就检查A是否对称，它的对角线元素是否都为正。如果这两个条件都成立，就试着进行Cholesky分解，并求三角伴随解。

3. 如果A是方阵，且前面的条件都不成立，那么就用LU分解和向后代入法解方程组。

4. 如果A不是方阵，就计算A的QR分解，并得出方程组的最小二乘解（见9.2.3节中关于QR分解如何应用到最小二乘问题的讨论）。

426

8.5 非线性方程组

对方程组$Ax = b$，如果A中系数或b中元素依赖于一个或多个x_i，那么此方程组就是非线性的。对一个线性方程来说，A和b的元素可以在x未知的情况下求得。而对一个非线性方程来说，A或b（或A和b）只有在x已知时才是已知的。

例8.10 直线与抛物线的交点

考虑在(x，y)平面上定义的一条直线和一条抛物线构成的2 × 2方程组:

$$y = \alpha x + \beta$$
$$y = x^2 + \sigma x + \tau$$

其中α、β、σ和τ是直线和抛物线的参数。如果两条曲线存在交点，那么这些交点就是此方程组的解。将前面的方程写成方程（8-52）的形式，令$x_1 = x$，$x_2 = y$，方程组变成：

$$\alpha x_1 - x_2 = \beta$$

$$(x_1+\sigma)x_1 - x_2 = -\tau$$

用矩阵符号表示为：

$$\begin{bmatrix} \alpha & -1 \\ x_1+\sigma & -1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -\beta \\ -\tau \end{bmatrix} \tag{8-50}$$

对于此问题，只有系数矩阵的元素a_{21}会导致非线性的产生。在例8.12和例8.13中给出了方程（8-50）的解。

要注意，方程（8-50）并非是定义此方程组的A和b的惟一途径（见练习44）。

8.5.1 用迭代法求解非线性系统

一个线性系统的形式解是$x = A^{-1}b$。既然右边与x无关，那么线性方程组的解就可以使用所谓的直接法来得到。给定一个矩阵和已知维数的右边向量，用直接法如向后代入的高斯消去法，就可以经过已知的有限步骤操作来结束求解（参考表8-1）。本文中，"直接"表示求解过程有限的特性。对一个非线性方程组来说，表达式$x = A^{-1}(x)\ b(x)$表明等式右边在x未知的情况下不能进行计算。最好是设计一种方法，给出一系列越来越能满足$A(x)\ x = b(x)$的x猜测值。这需要与直接法相反的迭代法。理想状态下，迭代过程会产生越来越接近真实值的解。当解"足够接近"时，迭代过程就中止。原则上，要使用迭代法得到精确解，需要无限次数的迭代步。 427

一个非线性方程组可以写成我们熟悉的形式：

$$Ax = b \tag{8-51}$$

但是现在$A = A(x)$，$b = b(x)$。非线性方程组的求解方法是形式为$f(\xi) = 0$的标量方程求根法的一般化（见第6章）。解非线性方程组时，ξ就变成了一个未知向量，解$Ax = b$就等价于求解x使得方程(8-52)成立。

$$f(x) = Ax - b = 0 \tag{8-52}$$

其中$f(x)$是x的向量值函数（vector valued function）。另外，也可以将非线性问题表示成残差

$$r = b - Ax = -f(x) \tag{8-53}$$

用迭代法求解方程（8-51）可以写成

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}, \quad k=1,2,\ldots \tag{8-54}$$

其中迭代计数器k周围的圆括号表明$x^{(k)}$并非是x的k次方。要使用此公式计算，需要首先由线性化的系数矩阵和右边向量来计算修正向量（update vector）$\Delta x^{(k)}$，

$$A^{(k)} = A(x^{(k)}), \qquad b^{(k)} = b(x^{(k)})$$

在计算修正向量之前，向量$f^{(k)} = f(x^{(k)})$或

$$f^{(k)} = A^{(k)}x^{(k)} - b^{(k)} \tag{8-55}$$

不能为零，除非$x^{(k)}$为此非线性问题的解。因此，通过检查$\|f^{(k)}\|$可以监控迭代法的收敛性。算法8.10中包含了迭代求解方程（8-51）的逻辑。

算法8.10　非线性方程组的迭代求解法

```
initialize: x=x^(0)
for k=0,1,2,...
    A^(k)=A(x^(k))                  (线性化A)
    b^(k)=b(x^(k))                  (线性化b)
    f^(k)=A^(k)x^(k) - b^(k)
    if ||f^(k)|| is small enough, stop
    Δx^(k+1)=...                    (计算修正向量)
    x^(k+1)=x^(k)+Δx^(k)
end
```

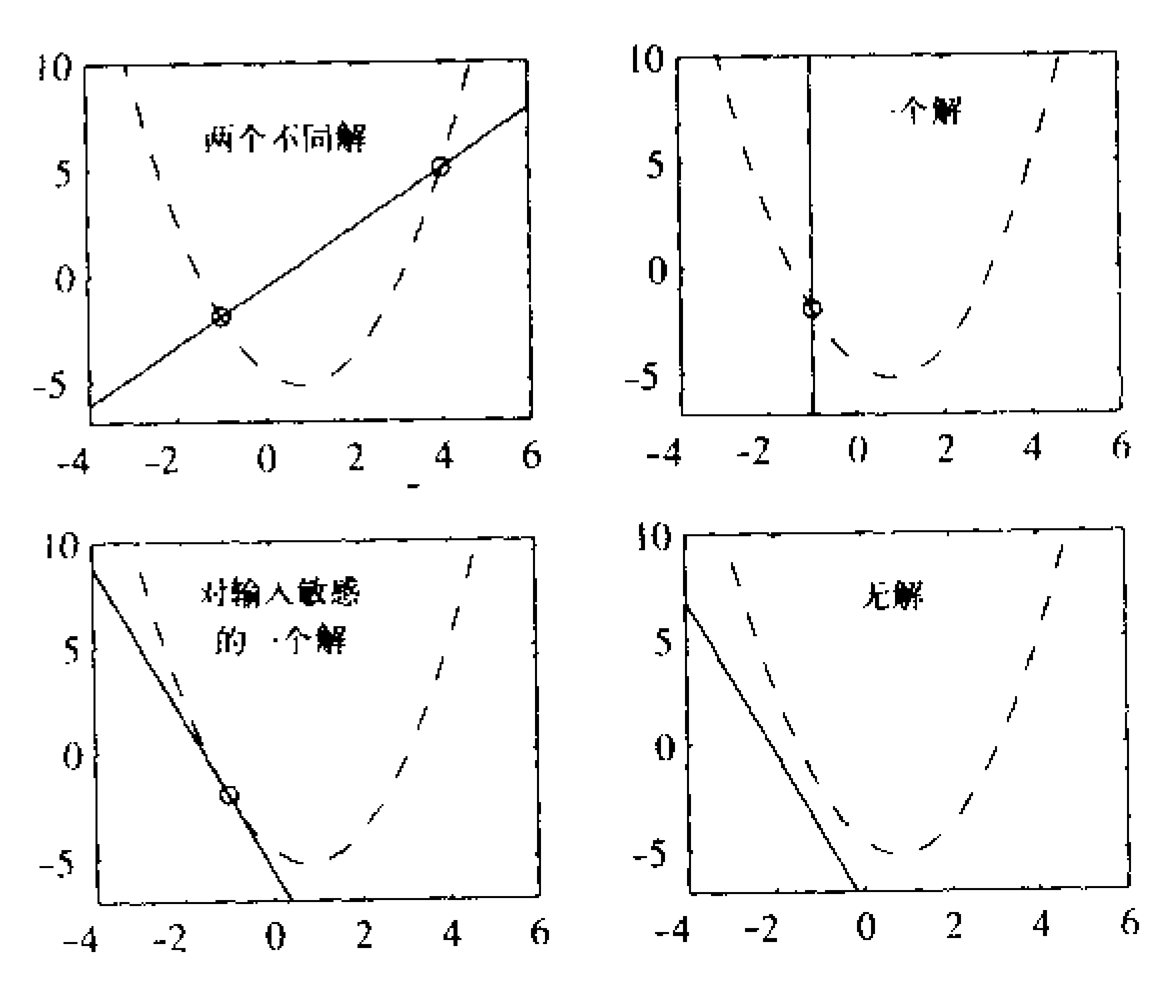
428

计算修正向量的两个程序在后面给出。

除了不能在有限步骤内求解出方程（8-51）的解外，方程组的非线性特征还带来了其他的一些数学复杂性问题，其中最重要的就是方程（8-51）或与之等价的方程（8-52）的解不能保证是惟一的，即使用数值方法得到一个解，也不能肯定它就是方程组的惟一解。

例8.11　解的类型

例8.10推导出了一条直线和一条抛物线（二次曲线）相交的方程组。这些方程有简单的几何意义，可以描述出可能存在的解的类型。图8-9描述了此方程组的四种可能的结果。这里可能有0个、1个或2个解，这取决于α、β、σ和τ的值。对图8-9中的曲线，只有α和β改变。



429

图8-9　例8.10中的非线性方程组的解的情况

8.5.2 逐次代换法

逐次代换法（successive substitution）是解非线性方程组的一种简单迭代方法。在解法的每一步，都会求解线性方程组$A^{(k)}x^{(k+1)}=b^{(k)}$以得到解的一个新的猜测值。

算法8.11 逐次代换法

initialize: $x=x^{(0)}$
for $k=0, 1, 2, \ldots$
  $A^{(k)}=A(x^{(k)})$
  $b^{(k)}=b(x^{(k)})$
  $f^{(k)}=A^{(k)}x^{(k)}-b^{(k)}$
  if $\|f^{(k)}\|$ is small enough, stop
  solve $A^{(k)}x^{(k+1)}=b^{(k)}$
end

逐次代换法对非线性化不高的问题有效。注意，修正向量$\Delta x^{(k)}$在算法8.11中并未出现。修正也可以明确地表示如下，将方程（8-54）代入$A^{(k)}x^{(k+1)}=b^{(k)}$得

$$A^{(k)}(x^{(k)}+\Delta x^{(k)})=b^{(k)}$$

整理后得

$$A^{(k)}\Delta x^{(k)}=-f^{(k)}$$

因此

$$A^{(k)}x^{(k+1)}=b^{(k)}$$

就等价于下列两个步骤

$$A^{(k)}\Delta x^{(k)}=-f^{(k)}$$
$$x^{(k+1)}=x^{(k)}+\Delta x^{(k)}$$

例8.12 使用逐次代换法求解2 × 2方程组

使用逐次代换法求出由

$$y=1.4x-0.6$$
$$y=x^2-1.6x-4.6$$

定义的直线和二次方程的交点。此方程组写成矩阵形式为：

$$\begin{bmatrix}1.4 & -1\\ x_1-1.6 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}=\begin{bmatrix}0.6\\ 4.6\end{bmatrix}$$

其中$x_1=x$，$x_2=y$。此方程组等价于$\alpha=1.4$，$\beta=0.6$，$\sigma=-1.6$和$\tau=-4.6$的方程（8-50）。在这些参数下，方程组有$(x, y)=(-1, -2)$和$(x, y)=(4, 5)$两个解，如图8-9的左上图所示。程序清单8-7中的demoSSub函数实现了逐次代换法。用户可以提供可选的迭代总数maxit和解的初始猜测值x_0。 430

令初始猜测值为缺省的$x_0=[0, 0]^T$，运行demoSSub作10次迭代，得：

```
>> x = demoSSub(10);
```

```
   k       x(1)       x(2)       norm(f)
   1    -1.33333   -2.46667    4.64e+00
   2    -0.92308   -1.89231    1.78e+00
   3    -1.01961   -2.02745    3.79e-01
   4    -0.99512   -1.99317    9.84e-02
   5    -1.00122   -2.00171    2.44e-02
   6    -0.99969   -1.99957    6.11e-03
   7    -1.00008   -2.00011    1.53e-03
   8    -0.99998   -1.99997    3.81e-04
   9    -1.00000   -2.00001    9.54e-05
  10    -1.00000   -2.00000    2.38e-05
```

程序清单8-7　函数demoSSub示范了用逐次代换法求解非线性方程组

```
function x = demoSSub(maxit,x0)
% demoSSub  Solve a 2-by-2 nonlinear system by successive substitution
%           The system is
%                     1.4*x1 - x2 = 0.6
%              x1^2 - 1.6*x1 - x2 = 4.6
%
% Synopsis:  x = demoSSub(maxit,x0)
%
% Input:   maxit = (optional) max number of iterations.  Default: maxit = 5
%          x0 = (optional) initial guess at solution.  Default:  x0 = [0; 0]
%
% Output:  x = estimate of solution after maxit iterations

if nargin<1,  maxit=5;           end
if nargin<2,  x0 = zeros(2,1);   end

% --- Coefficients for the case of two distinct solutions
alpha =  1.4;  bbeta = -0.6;  sigma = -1.6;  tau = -4.6;

b = [-bbeta; -tau];
x = x0;

fprintf('\n   k       x(1)       x(2)       norm(f)\n');
for k = 1:maxit
   A = [ alpha -1; (x(1)+sigma) -1];
   f = A*x - b;
   x = A\b;
   fprintf('%4d   %9.5f   %9.5f   %10.2e\n',k,x(1),x(2),norm(f));
end
```

431

如果存在多个解，逐次代换法就会接近一个解而远离其他的解，例如，即使初始猜测值接近解$(x, y) = (4, 5)$，逐次代换法仍然会收敛于解$(x, y) = (-1, -2)$。如果初始猜测值为$x_0 = [5, 5]^T$和$x_0 = [6, 6]^T$，情况也与此类似。

```
>> x = demoSSub(10,[5;5]);

   k       x(1)       x(2)       norm(f)
   1     2.00000    2.20000    7.53e+00
   2    -4.00000   -6.20000    6.00e+00
```

```
  10     -1.00003     -2.00004      5.72e-04

>> x = demoSSub(10,[6;6]);

   k        x(1)         x(2)        norm(f)
   1      1.33333      1.26667      1.59e+01
   2     -2.40000     -3.96000      6.22e+00
   .
  10     -1.00002     -2.00002      3.34e-04
```

8.5.3 牛顿法

解方程组的牛顿法类似于解标量方程的牛顿法。只有当

$$f(x)=\begin{bmatrix} f_1(x_1,x_2,\ldots,x_n)) \\ f_2(x_1,x_2,\ldots,x_n)) \\ \vdots \\ f_n(x_1,x_2,\ldots,x_n)) \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

成立时，才可以得到$n\times n$非线性方程组的解。与前面相同，令$x^{(k)}$为第k次迭代的解的猜测值。假设$\|f^{(k)}\|$不是足够小，我们找一个修正向量$\Delta x^{(k)}$: 432

$$x^{(k+1)}=x^{(k)}+\Delta x^{(k)}\Leftrightarrow\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix}=\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}+\begin{bmatrix} \Delta x_1^{(k)} \\ \Delta x_2^{(k)} \\ \vdots \\ \Delta x_n^{(k)} \end{bmatrix}$$

以使$f(x^{(k+1)})=0$。使用Taylor定理的多维扩展来近似$f(x)$在$x^{(k)}$的邻域内的变化，得

$$f(x^{(k)}+\Delta x^{(k)})=f(x^{(k)})+f'(x^{(k)})\Delta x^{(k)}+O\left(\|\Delta x^{(k)}\|^2\right) \tag{8-56}$$

其中$f'(x^{(k)})$是方程组的雅可比（Jacobian）矩阵：

$$f'(x)\equiv J(x)=\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & \ddots & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{8-57}$$

忽略高次项，令$J^{(k)}$为$x^{(k)}$处的雅可比值。整理方程（8-56）得

$$J^{(k)}\Delta x^{(k)}=-f(x^{(k)})+f(x^{(k)}+\Delta x^{(k)})$$

Newton迭代法的目标是使$f(x^{(k)}+\Delta x^{(k)})=0$，故将前面方程的$f(x^{(k)}+\Delta x^{(k)})$项设为零，得

$$J^{(k)}\Delta x^{(k)}=-f(x^{(k)}) \tag{8-58}$$

方程（8-58）是一个关于n个未知的$\Delta x^{(k)}$和n个线性方程构成的方程组。虽然一般来说J和f的元素都依赖于x，但是方程组已经通过在$x^{(k)}$处计算J和f而线性化了。每一步牛顿迭代都要计算向

[433] 量$f^{(k)}$、矩阵$J^{(k)}$，并求解方程（8-58）的解。算法8.12实现了非线性方程组的牛顿迭代法。

算法8.12 牛顿法解方程组

initialize: $x = x^{(0)}$
for $k = 0, 1, 2, \ldots$
  $f^{(k)} = A^{(k)}x^{(k)} - b^{(k)}$
  if $\|f^{(k)}\|$ is small enough, stop
  compute $J^{(k)}$
  solve $J^{(k)}\Delta x^{(k)} = -f^{(k)}$
  $x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$
end

如例8.13所示，在求解$f^{(k)}$时，不需要明确地求解出$A^{(k)}$。

例8.13 使用牛顿法求解2×2方程

例8.12示范了如何使用逐次代换法来求解方程（8-50）中的2×2非线性方程组。这里，我们将使用牛顿法来求解此问题。首先，定义向量f:

$$f = \begin{bmatrix} \alpha x_1 - x_2 + \beta \\ x_1^2 + \sigma x_1 - x_2 + \tau \end{bmatrix}$$

雅可比矩阵中的各项为:

$$\frac{\partial f_1}{\partial x_1} = \alpha, \qquad \frac{\partial f_1}{\partial x_2} = -1$$

$$\frac{\partial f_2}{\partial x_1} = 2x_1 + \sigma, \qquad \frac{\partial f_2}{\partial x_2} = -1$$

因此

$$J = \begin{bmatrix} \alpha & -1 \\ (2x_1 + \sigma) & -1 \end{bmatrix}$$

程序清单8-8中的demoNewtonSys函数实现了用牛顿法求解此2×2方程组。在缺省输入参数的情况下运行demoNewtonSys函数，得:

```
>> x = demoNewtonSys;

 k        x(1)        x(2)       norm(f)       norm(dx)
 0      0.00000     0.00000     4.64e+00      2.80e+00
 1     -1.33333    -2.46667     1.78e+00      5.40e-01
 2     -1.01961    -2.02745     9.84e-02      3.36e-02
 3     -1.00008    -2.00011     3.81e-04      1.31e-04
 4     -1.00000    -2.00000     5.82e-09      2.00e-09
 5     -1.00000    -2.00000
```

[434] 使用Newton法只需4步迭代得到的解就比使用逐次代换法进行10步迭代得到的解精确得多。若给定合适的初值，经过几步也可以得到另一个解:

```
>> x = demoNewtonSys(5,[5; 5]);

 k        x(1)        x(2)       norm(f)       norm(dx)
```

0	5.00000	5.00000	7.53e+00	8.80e-01
1	4.14286	5.20000	7.35e-01	2.39e-01
2	4.00386	5.00541	1.93e-02	6.64e-03
3	4.00000	5.00000	1.49e-05	5.12e-06
4	4.00000	5.00000	8.86e-12	3.05e-12
5	4.00000	5.00000		

程序清单8-8 函数demoNewtonSys使用牛顿法来求解由两个非线性方程所组成的方程组

```
function x = demoNewtonSys(maxit,x0)
% demoNewtonSys   Solve a 2-by-2 nonlinear system by Newton's method
%                 The system is
%                             1.4*x1 - x2 = 0.6
%                     x1^2 - 1.6*x1 - x2 = 4.6
%
% Synopsis:  x = demoNewtonSys(maxit,x0)
%
% Input:    maxit = (optional) max number of iterations.  Default: maxit = 5
%           x0 = (optional) initial guess at solution.  Default:  x0 = [0; 0]
%
% Output:  x = estimate of solution after maxit iterations

if nargin<1,  maxit=5;          end
if nargin<2,  x0 = zeros(2,1);  end

% --- Coefficients for the case of two distinct solutions
alpha =  1.4;  bbeta = -0.6;  sigma = -1.6;  tau = -4.6;

x = x0;   f = zeros(size(x));

fprintf('\n   k       x(1)        x(2)      norm(f)      norm(dx)\n');
for k = 1:maxit
   f(1) = alpha*x(1) - x(2) + bbeta;
   f(2) = x(1)^2 + sigma*x(1) - x(2) + tau;
   J = [ alpha  -1; (2*x(1)+sigma)  -1 ];
   dx = -J\f;
   fprintf('%4d    %9.5f    %9.5f    %10.2e    %10.2e\n',...
           k-1,x(1),x(2),norm(f),norm(dx));
   x = x + dx;
end
fprintf('%4d    %9.5f    %9.5f\n',k,x(1),x(2));
```

435

Newton法的一般实现 函数demoNewtonSys示范了如何使用牛顿法来求解两个联立的非线性方程。牛顿法的一般实现是将求解算法从J和f的求值中分离出来。程序清单8-9的newtonSys函数就是Newton法的一般实现。细心的读者会注意到，在6.4节描述的非线性标量方程的求解过程中使用的newton函数与newtonSys函数非常相似。

调用函数newtonSys时，能够使用多个可选的输入参数。其中最简单的用法为：

```
x = newtonSys('Jfun', x0)
```

其中，*Jfun*是在特定问题中计算J和f的m文件的名称，x0是解的初始猜测值。例8.14示范了如何使用newtonSys函数的可选输入变量来求解实际工程问题。

函数newtonSys对给定方程组进行牛顿迭代，直到满足下面两个收敛条件中的一个为止。

$$\|f\|_2 < \delta_f, \qquad \|\Delta x\|_2 < \delta_x$$

牛顿迭代法的求解过程可以通过将可选的输入参数verbose设置为非零值来进行监测。

程序清单8-9 函数newtonSys使用牛顿法来求解一个用户自定义的方程组

```
function x = newtonSys(Jfun,x0,xtol,ftol,maxit,verbose,varargin)
% newtonSys  Newton's method for systems of nonlinear equations.
%
% Synopsis:  x = newtonSys(Jfun,x0)
%            x = newtonSys(Jfun,x0,xtol)
%            x = newtonSys(Jfun,x0,xtol,ftol)
%            x = newtonSys(Jfun,x0,xtol,ftol,verbose)
%            x = newtonSys(Jfun,x0,xtol,ftol,verbose,arg1,arg2,...)
%
% Input:  Jfun = (string) name of mfile that returns matrix J and vector f
%         x0   = initial guess at solution vector, x
%         xtol = (optional) tolerance on norm(x).  Default: xtol=5e-5
%         ftol = (optional) tolerance on norm(f).  Default: ftol=5e-5
%         verbose = (optional) flag.  Default: verbose=0, no printing.
%         arg1,arg2,... = (optional) optional arguments that are passed
%                   through to the mfile defined by the 'Jfun' argument
%
% Note:  Use [] to request default value of an optional input.  For example,
%        x = newtonSys('JFun',x0,[],[],[],arg1,arg2) passes arg1 and arg2 to
%        'JFun', while using the default values for xtol, ftol, and verbose
%
% Output:  x = solution vector;  x is returned after k iterations if tolerances
%              are met, or after maxit iterations if tolerances are not met.

if nargin < 3 | isempty(xtol),    xtol = 5e-5;  end
if nargin < 4 | isempty(ftol),    ftol = 5e-5;  end
if nargin < 5 | isempty(maxit),   maxit = 15;   end
if nargin < 5 | isempty(verbose), verbose = 0;  end
xeps = max(xtol,5*eps);  feps = max(ftol,5*eps);    %  Smallest tols are 5*eps

if verbose, fprintf('\nNewton iterations\n  k     norm(f)      norm(dx)\n'); end

x = x0;  k = 0;          %  Initial guess and current number of iterations
while k <= maxit
  k = k + 1;
  [J,f] = feval(Jfun,x,varargin{:});   %   Returns Jacobian matrix and f vector
  dx = J\f;
  x = x - dx;
  if verbose, fprintf('%3d %12.3e %12.3e\n',k,norm(f),norm(dx));  end
  if ( norm(f) < feps ) | ( norm(dx) < xeps ),  return;   end
end
warning(sprintf('Solution not found within tolerance after %d iterations\n',k));
```

436
437

例8.14 三水槽中水的分配问题

在我们生活中的很多方面，管道网络都很重要。城镇的供水系统就是一个管道网络，在工业生产中压缩空气、冷却水和处理化学制品的管道系统也都是管道网络。分析管道网络可

以得到非线性方程组。

管道网络的一个简单例子是图8-10中所描述的经典的三水槽问题。很多较大管道网络问题都有该系统的影子。三个管道连接三个水槽并汇于一个公共结点。这些管道有不同的长度和直径，三个水槽也在不同的高度上。已知这些物理参数，我们希望知道管道中的水流速。注意，通过调整水槽的高度，可以控制水流速的大小和方向。

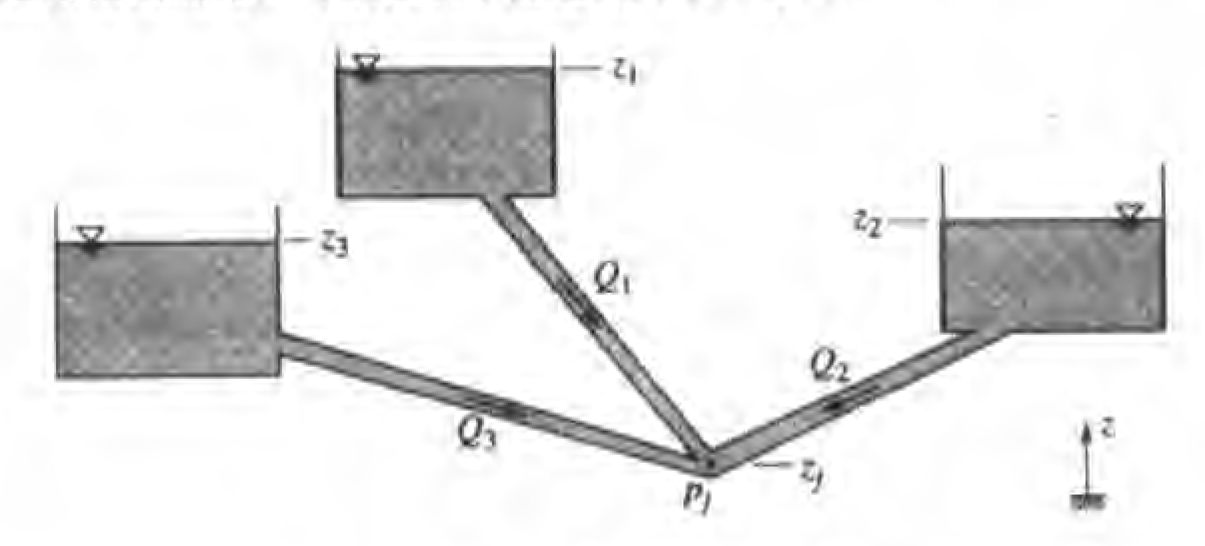


图8-10 三水槽的水的分配问题

由质量和能量守恒定律，我们可以得到描述水流速的非线性方程组（见F.M. White, *Fluid Mechanics*, 4th ed., 1999,McGraw-Hill, New York）。由结点处的质量守恒[⊖]得

$$Q_1 + Q_2 + Q_3 = 0 \tag{8-59}$$

其中Q_i为管道中体积流量(m^3/s或ft^3/s)。方程（8-59）中默许一个约定，就是流向结点的流率为正的。另外，我们知道体积流量也与流体的流动速度有关，即$Q_i = V_iA_i$，其中V_i是平均流动速度，A_i是管道横截面的面积。

在管道的入口和出口处，根据能量守恒定律得到的方程为：

$$\left[\frac{p}{\gamma}+\frac{V^2}{2g}+z\right]_{in} = \left[\frac{p}{\gamma}+\frac{V^2}{2g}+z\right]_{out} + h_f \tag{8-60}$$

其中p是流体压力，γ是流体的比重，V是管道中的平均流动速度，g是重力加速度，z是水面相对于参考平面的高度，h_f是管道中的水压摩擦损失（frictional head loss）。

管道中的粘性水压损失用Darcy-Weisbach方程来计算：

$$h_f = K_f \frac{L}{d}\frac{V^2}{2g} \tag{8-61}$$

其中K_f是一个据经验得出的摩擦系数，L是管道长度，d是管道直径。要在本问题中应用方程（8-60），我们应该注意到因为水槽的表面暴露在大气中，测量气压为零标准计量压强，故$p_1 = p_2 = p_3 = 0$。在本问题中，可以忽略流体的动能的改变[⊜]，因此方程（8-60）中的V^2项可以忽略。将方程（8-60）和方程（8-61）分别应用于三段管道中，得

438

⊖ 该方程类似于电流的基尔霍夫（Kirchhoff）定律。

⊜ 更仔细的分析，就要计算每个水槽的入口（或出口）处微小的动能损失，要对Q_i和P_j进行一定的修正，但这里为了表述清晰，忽略了动能损失。

$$\frac{p_j}{\gamma}+c_1Q_1^2+z_j-z_1=0$$

$$\frac{p_j}{\gamma}+c_2Q_2^2+z_j-z_2=0$$

$$\frac{p_j}{\gamma}+c_3Q_3^2+z_j-z_3=0$$

其中

$$c_i=K_{f,i}\frac{L_i}{d_i}\frac{1}{2gA_i^2}$$

对每一段管道都是常数（假设$K_{f,i}$与V_i无关）。

此问题的未知数是三个流量及结点处的压力。向量x和f为：

$$x=\begin{bmatrix}Q_1\\Q_2\\Q_3\\p_j\end{bmatrix},\qquad f=\begin{bmatrix}Q_1+Q_2+Q_3\\p_j/\gamma+c_1Q_1^2+z_j-z_1\\p_j/\gamma+c_2Q_2^2+z_j-z_2\\p_j/\gamma+c_3Q_3^2+z_j-z_3\end{bmatrix}$$

计算雅可比矩阵中的各项，得：

$$\frac{\partial f_1}{x_1}=1,\quad \frac{\partial f_1}{x_2}=1,\quad \frac{\partial f_1}{x_3}=1,\quad \frac{\partial f_1}{x_4}=0$$

$$\frac{\partial f_2}{x_1}=2c_1Q_1,\quad \frac{\partial f_2}{x_2}=0,\quad \frac{\partial f_2}{x_3}=0,\quad \frac{\partial f_2}{x_4}=\frac{1}{\gamma}$$

$$\frac{\partial f_3}{x_1}=0,\quad \frac{\partial f_3}{x_2}=2c_2Q_2,\quad \frac{\partial f_3}{x_3}=0,\quad \frac{\partial f_3}{x_4}=\frac{1}{\gamma}$$

$$\frac{\partial f_4}{x_1}=0,\quad \frac{\partial f_4}{x_2}=0,\quad \frac{\partial f_4}{x_3}=2c_3Q_3,\quad \frac{\partial f_4}{x_4}=\frac{1}{\gamma}$$

因此，

$$J=\begin{bmatrix}1&1&1&0\\2c_1Q_1&0&0&1/\gamma\\0&2c_2Q_2&0&1/\gamma\\0&0&2c_3Q_3&1/\gamma\end{bmatrix}\tag{8-62}$$

前面求出的f公式并不能适用于任何高度的水槽。只要写出恒定横截面（$V_1=V_2$）的水平 439 管道($z_1=z_2$)的能量方程，该公式的问题就可以暴露出来：

$$\frac{p_1-p_2}{\gamma}=cQ^2$$

由经验可知，如果流体由位置1流向位置2（即$Q>0$），那么就有$p_1>p_2$。如果流体流向相反（即$Q<0$），那么就有$p_1<p_2$。如果我们使用

$$\frac{p_1-p_2}{\gamma}=-\mathrm{sign}(Q)cQ^2$$

就可以将这层意思恢复到能量公式中。此修改应用于f，得到：

$$f=\begin{bmatrix} Q_1+Q_2-Q_3 \\ p_j/\gamma+c_1\,\mathrm{sign}(Q_1)Q_1^2+z_j-z_1 \\ p_j/\gamma+c_2\,\mathrm{sign}(Q_2)Q_2^2+z_j-z_2 \\ p_j/\gamma-c_3\,\mathrm{sign}(Q_3)Q_3^2+z_j-z_3 \end{bmatrix} \tag{8-63}$$

方程（8-62）中定义的J和方程（8-63）中定义的f编码在程序清单8-11的JfReservoir函数中。其主程序是程序清单8-10中的demoThreeRes函数。demoThreeRes函数定义了一组水槽和连接管道的L、d和z的值，然后，通过设置初始猜测值和对函数newtonSys的调用来求解方程组。

在缺省参数下运行demoThreeRes，得：

```
>> demoThreeRes

Newton iterations
  k     norm(f)       norm(dx)
  1    7.581e+01     1.400e+05
  2    3.734e+01     1.082e+05
  3    1.795e+02     4.277e+04
  4    4.333e+01     4.365e+04
  5    6.942e+00     1.929e+03
  6    4.972e-01     2.465e+02
  7    3.376e-03     1.865e+00
  8    1.586e-07     8.801e-05

Flow rates: Q1  = 1.19, Q2 = -0.32, Q3 = -0.86 m^3/s
Junction pressure   = 134841 Pa
Net flow rate into junction    = 0.000e+00 m^3/s
```

由于缺省的高度为z_1 = 30，z_2 = 18，z_3 = 9以及z_j = 11，水会从水槽1流向水槽2和水槽3。440 建议读者实验不同的水槽高度。此模型能得到demoThreeRes([12;14;16;5])和demoThreeRes([12;12;12;15])的预期结果吗？

程序清单8-10 函数demoThreeRes使用newtonSys来求解三水槽问题中的非线性方程组

```
function demoThreeRes(z,maxit)
% demoThreeRes  Solve the three reservoir problem with Newton's method
%
% Synopsis:  demoThreeRes(z,maxit)
%
% Input:  z = (optional) vector of four elements containing the elevation
%             of each of the three reservoirs (z(1:3)) and the elevation
%             of the junction, z(4).  Default:  z = [30; 18; 9; 11] (m).
%         maxit = (optional) max number of iterations.  Default: maxit = 5
%
% Output:  Solution is printed to command window

if nargin<1,  z = [30; 18; 9; 11];  end
if nargin<2,  maxit=15;             end

% --- Assign problem parameters
L = [3000; 600; 1000];            %  Lengths of pipes, (m)
```

```
d = [1; 0.45; 0.6];                       % diameters of pipes, (m)
area = 0.25*pi*d.^2;                      % area of pipe sections
Kf = [0.015; 0.024; 0.020];               % friction factor, assumed constant
c = Kf.*(L./d)./(2*9.8*area.^2);          % constant in head loss formula
q  = [1; 1; 1];                           % intial guess at flow rates, (m^3/s)
pj = 1e5;                                 % initial guess at junction pressure, (Pa)

% --- Solve the system and print results
x0 = [q; pj];                             % initial guess at solution vector
x = newtonSys('JfReservoir',x0,[],[],maxit,1,z,c);

fprintf('\nFlows rates. Q1 = %5.2f, Q2 = %5.2f, Q3 = %5.2f m^3/s\n',x(1:3));
fprintf('Junction pressure = %8.0f Pa\n',x(4));
fprintf('Net flow rate into junction = %12.3e m^3/s\n',sum(x(1:3)));
```

441

程序清单8-11 函数JfReservoir用来计算三水槽问题的雅可比矩阵和右边向量

```
function [J,f] = JfReservoir(x,z,c)
% JFreservoir  Jacobian and f vector for three reservoir system
%
% Synopsis:  [J,f] = JFreservoir(x,z,c)
%
% Input: x = current guess at solution vector
%        z = vector of elevations of reservoirs and junction, (m)
%            z = [z1; z2; z3; zj]
%        c = constant in Darcy-Weisbach equation, c = f (L/d)(V^2/2/g)(1/A)
%            for each pipe.
%
% Output: J = Jacobian matrix for the system
%         f = right hand side vector for Newton iterations

gamw = 9790;                              % specific weight of water, (N/m^3)
q = x(1:3);    pj = x(4);    zj = z(4);   % extract problem variables from x
f = zeros(4,1);    J = zeros(4,4);        % Initialize

% ---- equation 1.  sum(q) = 0
f(1) = q(1) + q(2) + q(3);
J(1,1) = 1;  J(1,2) = 1;  J(1,3) = 1;

%  -- equation 2:  head loss in branch 1
f(2) = pj/gamw + sign(q(1))*c(1)*q(1)^2 + zj - z(1);
J(2,1) = sign(q(1))*2*c(1)*q(1);
J(2,4) = 1/gamw;

% --- equation 3:  head loss in branch 2
f(3) = pj/gamw + sign(q(2))*c(2)*q(2)^2 + zj - z(2);
J(3,2) = sign(q(2))*2*c(2)*q(2);
J(3,4) = 1/gamw;

% --- equation 4:  head loss in branch 3
f(4) = pj/gamw + sign(q(3))*c(3)*q(3)^2 + zj - z(3);
J(4,3) = sign(q(3))*2*c(3)*q(3);
J(4,4) = 1/gamw;
```

442

8.6 小结

方程组$Ax=b$有解当且仅当A和b是相容的（即当且仅当b在A的列空间里）。与此等价，如果$\text{rank}(A)=\text{rank}([A\ b])$，$A$和$b$就是相容的。另外，方程组$Ax=b$的解（如果存在）是惟一的当且仅当$\text{rank}(A)=n$。对任意有$n$个方程和$n$个未知数的方程组（即$A$是$n\times n$方阵）来说，如果有$\text{rank}(A)=n$，那么它就是相容的。因此，$n\times n$方程组的解存在且惟一的条件是当且仅当$\text{rank}(A)=n$。

如果$n\times n$矩阵A秩为n，那它就是非奇异的和可逆的。若A可逆，那么其逆矩阵A^{-1}存在，且$Ax=b$的形式解为$x=A^{-1}b$。一般地，在解$Ax=b$时，并不要先计算出A^{-1}再计算$A^{-1}b$。数学表达式$x=A^{-1}b$应该解释成"使用合适的数值技术方法来解$Ax=b$，如采用向后代入策略的高斯消去法"。

采用向后代入策略的高斯消去法等价于通过求系数矩阵的LU分解并由L和U因子的三角法来解方程组。如果系数矩阵是对称正定的，就可以使用Cholesky分解法。使用Cholesky分解来解方程组可以将浮点操作次数降低一半。

运算符\是MATLAB中求解线性方程组的一种较好的方法。它使用不同的求解方法，这些方法依赖于系数矩阵的形状和其他一些特性。参考8.4.3节中有关运算符\如何选择解法的总结。

$Ax=b$的数值解$\hat{x}$与其真实解x不同，因为在求解过程的所有步骤中都含有舍入误差。舍入误差最开始是在A和b存入计算机内存时产生的。求解$\hat{x}$的过程中，即在矩阵A的消去（或称为分解）和向后代入处理中，会产生更多的舍入误差。A和b的扰动对数值解的影响程度取决于条件数$\kappa(x)$的大小。较大的$\kappa(x)$值表明A接近奇异，这时A就被认为是病态的（条件不足），A和b中较小的扰动都会带来$\hat{x}$与x间相对巨大的差异。

如果$\kappa(x)$很小，就可以使用较稳定的算法，如带局部选主元的高斯消去法，求解出有较小残差的解，残差为$r=b-A\hat{x}$。$\hat{x}$中正确的有效位数可以由方程（8-38）估算出来。如果$\kappa(x)$较大，用稳定的算法来进行计算，即使返回结果的残差很小，所得的解也可能有很大的误差。

非线性方程组需要使用迭代法来求解，因为A或b（或A和b）的值都依赖于解向量x。迭代中每一步的计算量，都至少相当于解一个同样大小的线性方程组。8.5节中讲解了逐次代换法和牛顿法。NMM工具箱中的newtonSys函数提供了用牛顿法求解非线性方程组的一般实现。 443

表8-3列出了本章编写的m文件。

表8-3 实现方程组的各种解法的NMM工具箱函数。这些m文件函数包含在NMM工具箱的LINALG目录下

函数	小节	描述
Cholesky	8.4.2	对称正定矩阵的Cholesky分解
demoNewtonSys	8.4.1	使用牛顿法求解一个2×2非线性方程组
demoSSub	8.5.2	使用逐次代换法求解一个2×2非线性方程组
demoThreeRes	8.5.3	使用牛顿法求解一个在三水槽问题中得出的含4个方程的非线性方程组
GEPivShow	8.2.4	使用高斯消去法求解一个方程组并打印出每一步消去的结果，采用局部选主元策略
GEshow	8.2.4	使用高斯消去法求解一个方程组并打印出每一步消去的结果，不采用选主元策略
JfReservoir	8.5.3	在三水槽问题中计算雅可比矩阵J和向量f的函数。见demoThreeRes和newtonSys。

（续）

函数	小节	描述
pumpCurve	8.2.5	解一个离心泵二次模型的方程组
luNopiv	8.4.1	不采用选主元策略的LU分解
luPiv	8.4.1	采用局部选主元策略的LU分解
newtonSys	8.5.3	用牛顿法求解非线性方程组的一般实现

补充读物

Strang [72]给出了求解方程组$Ax = b$基础知识的精彩描述。另外，他还提供了LU分解和向后代入法的MATLAB函数。Strang的程序简单易懂，并揭露出隐藏在MATLAB内置lu函数和反斜杠运算符中的一些细节。Datta [12]和Van Loan [77]也提供了高斯消去法和LU分解的MATLAB实现。

Stewart [68]对求解线性方程组的数值解作了很好的介绍，他一流的描述提供了更深入学习的桥梁。Gill等人在文献[28]中也对求解线性方程组作了全面而简洁的论述。Watkins [78]在线性方程组求解的计算方面有优秀的阐述。Kahaner 等人在文献[43]中描述了如何使用Fortran软件库来求解线性方程组，这个库是MATLAB第五版及以前版本使用的代码的基础。

444

Golub和Van Loan在文献[32]中对$Ax = b$的求解作了综合分析。另外，一些新近出版的书也在一个更高的层次上提供了数值线性代数的精妙分析。Demmel [15]描述了数值算法的当前发展形势。Trefethen和Bau在文献[76]中对数值计算的数学方面作了精妙的阐述，而Higham在文献[36]中对求解线性方程组过程中的舍入误差作了彻底的讨论。Anderson 等人在文献[4]中建立了LAPACK库，收录了求解线性方程组的新近的Fortran代码包。MATLAB的第六版中并入了LAPACK库。

在求解非线性方程组的问题上，大多数基础的数值分析教科书都提供了Newton法的简单介绍。此外还有很多其他的方法，并且其中非线性方程组的问题与非线性最优化的问题交迭在一起。读者若想进一步了解最新的且中等水平的迭代方法，可以参考Kelley 在文献[44]中的介绍，其中还包含MATLAB代码。若想了解关于非线性问题和最优化的讨论，可以参考Dennis和Schanbel在文献[16]中和Kelley在文献[45]中的介绍。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1. (1)将下列关于未知数a、b和c的方程集合转化成矩阵形式。此方程组存在惟一解吗？如果存在，是什么？

$$a = 4(b-c)$$

$$\frac{b-a}{2c} = 1$$

$$a+b = c-2$$

2. (1) 将下列关于未知数a、b和c的方程集合转化成矩阵形式。此方程组存在惟一解吗？如果存在，是什么？

$$a = 4(b - c)$$
$$\frac{b-a}{2c} = 1$$
$$a - b = 2(1 - c)$$

3. (1)已知R_1 = 2°C/W，R_2 = 0.5°C/W，R_3 = 35°C/W，R_4 = 0.7°C/W，R_5 = 1°C/W，那么方程（8-2）是否有解？若有解，是否惟一？

445

4. (2)例8.2中，推导了含7个未知数和7个方程的方程组。“写出方程的自然形式”部分的最后一段是对$Q_5 = Q_2$这一代换做的注释（见第367页）。假设方程$Q_2 = (T_w - T_a)/R_5$写成方程$Q_5 = (T_w - T_a)/R_5$，那么新方程组中有多少个未知数？要定义一个方程个数与未知数个数相同的方程组，还需要几个额外方程？写出这个新方程组的矩阵形式。使用习题3中的数值，判断此方程组是否有解。如果解存在，那么是否惟一？

5. (1)已知

$$\boldsymbol{Q} = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix}$$

手动计算求出$\boldsymbol{Q}^T\boldsymbol{Q}$。指出$\boldsymbol{Q}$所具有的特殊属性。

6. * (1+)手动计算求出$\boldsymbol{QRx} = \boldsymbol{b}$中的$\boldsymbol{x}$，其中

$$\boldsymbol{Q} = \begin{bmatrix} 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 0 & 1 & 0 \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix}, \quad \boldsymbol{R} = \sqrt{2}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 2 \\ 2\sqrt{2} \\ 4 \end{bmatrix}$$

（提示：利用上题中确定的$\boldsymbol{Q}$的属性来计算。）

7. (1+)手动计算求出$\boldsymbol{CDx} = \boldsymbol{b}$中的$\boldsymbol{x}$，已知

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/4 & 1/2 & 1 \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

（提示：在进行代数计算之前，先研究$\boldsymbol{C}$和$\boldsymbol{D}$的结构。）

8. (1+)手动计算求出$\boldsymbol{CDEx} = \boldsymbol{b}$中的$\boldsymbol{x}$，已知

$$\boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & 2 & 1 \end{bmatrix}, \quad \boldsymbol{D} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \quad \boldsymbol{E} = \begin{bmatrix} 1 & 4 & 3 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 20 \\ 84 \\ 70 \end{bmatrix}$$

（提示：在进行代数计算之前，先研究$\boldsymbol{C}$、$\boldsymbol{D}$和$\boldsymbol{E}$的结构。）

9. (1)在8.1节的总结中，把“相容性和秩的含义”使用粗体圆点列表列出了。将所有的rank ($\boldsymbol{A}$)代换为$\boldsymbol{A}$中向量的线性无关的合适描述，重新叙述所列的文本。

10. (2)判断下面的已知方程组是否相容。对相容的方程组，给出它们的解。如果有多个解，至少求出一个以上的解。

446

(a) $\begin{bmatrix}1 & 0\\1 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}$，　(b) $\begin{bmatrix}1 & 1\\0 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}2\\1\end{bmatrix}$，

(c) $\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$，　(d) $\begin{bmatrix}1 & 0\\0 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}0\\0\end{bmatrix}$，

(e) $\begin{bmatrix}1 & 1\\1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}2\\1\\1\end{bmatrix}$，　(f) $\begin{bmatrix}1 & 1\\1 & 0\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}1\\-1\\-1\end{bmatrix}$，

(g) $\begin{bmatrix}1 & 2 & 0 & 2\\0 & 1 & 3 & 4\\1 & 0 & 1 & 2\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}4\\4\\4\end{bmatrix}$，　(h) $\begin{bmatrix}1 & 2 & -3 & 1\\0 & 1 & -2 & 1\\1 & 0 & 1 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}-1\\2\\1\end{bmatrix}$，

11. (2)使用高斯消去法手动求解例8.3中的两个方程组。写出二者的增广方程组，继续使用第一、二行进行消去计算得到三角（增广）矩阵。每个增广方程组在消去的最后一步的最后一个方程是什么？这些方程与一般意义上所说的相容性有什么关系？

12. * (2+)由GEshow函数的代码，编写一个GErect函数来只对矩形矩阵（$m\times n$）进行高斯消去（无向后代入）。函数GErect应该返回三角系数矩阵 $\tilde{A}$ 和对应的右边向量 $\tilde{b}$。使用GErect函数来求解习题11。

13. (2)使用上题编制的GErect函数对$Ax = b$进行高斯消去，其中A和b定义为：

$$A=\begin{bmatrix}1 & 1\\2 & 1\\3 & 1\end{bmatrix},\quad b=\begin{bmatrix}2\\1\\0\end{bmatrix}$$

由三角化的系数矩阵，能否判断方程组就是相容的（另见例8.3）？

14. (2)使用\运算符求解方程组$Ax=b$，其中A和b定义为

$$A=\begin{bmatrix}1 & 1\\2 & 1\\3 & 1\end{bmatrix},\quad b=\begin{bmatrix}2\\1\\0\end{bmatrix}$$

447

（另见例8.3）。为什么x(1)+1和x(2)-3不为零？

15. (2)假设奇异方程为（参考例8.4）：

$$\begin{bmatrix}2 & 1\\2 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix}=\begin{bmatrix}6\\6\end{bmatrix}$$

若A和b中元素的扰动如下，求出对应的四个解（设$\delta = 5\times 10^{-9}$）：

(a) $a_{21} = 2+\delta$

(b) $a_{22} = 1+\delta$

(c) $a_{11} = 2+\delta$，$b_1 = 6+\delta$

(d) $a_{21} = 2+\delta$，$b_2 = 6+\delta$

16. * (2)编写一个lsolve函数来求解$Ax=b$，其中A是一个下三角矩阵。将你得到的解与使用反斜杠运算符所得到的解进行比较，检验你的函数。

17. (2) 编写一个usolve函数来求解$Ax=b$，其中A是一个上三角矩阵。将你得到的解与使用反斜杠运算符所得到的解进行比较，检验你的函数。
18. (1)假设例8.2中IC器件的热流网模型中的电阻取练习3中的值，且T_a =25°C，Q_c = 10W，那么其中的温度和热流为多少？
19. (1+)使用MATLAB内置的 \ 运算符来解下列方程组：

$$x_1+2x_2-x_4=9$$
$$2x_1+3x_2-x_3=9$$
$$4x_2+2x_3-5x_4=26$$
$$5x_1+5x_2+2x_3-4x_4=32$$

20. (2)由定义

```
function x = safeSolve (A, b)
```

编写一个safeSolve函数，提供一个 \ 运算符解$n \times n$方程组的接口。此函数应该能够检查A和b的维数以确定其匹配情况，以避免误用 \ 运算符引起的意外后果（本题中"匹配"并非是指A和b的相容性或A的秩）。如果A和b不匹配，则打印出一条相应的错误信息；如果A和b匹配，则计算$x = A\backslash b$并返回x的值。至少对一个匹配方程组和两个不匹配方程组使用你的函数，验证其有效。

21. * (3)矩阵A与它的逆矩阵A^{-1}满足$AA^{-1} = I$。使用按列乘的矩阵-矩阵相乘（参照算法7.5）可知A^{-1}的第j列满足$Ax = e_{(j)}$，其中$e_{(j)}$是单位矩阵的第j列（例如，e_3 = [0, 0, 1, …, 0]^T）。通过解$Ax = e_{(j)}$，j = 1，…n，一次可以解出A^{-1}的一列。 448
    (a) 编写一个名字叫invByCol的函数，一次一列地计算$n \times n$矩阵的逆矩阵。使用反斜杠运算符来求解A^{-1}的每一列。
    (b) 使用表8-1中的估算办法来估算函数invByCol计算$n \times n$矩阵时与n相关的浮点操作次数的数量级。
    (c) 对逐渐增大的矩阵测量invByCol函数的浮点操作次数，用来验证(b)中所推导出的估算是否正确。使用语句A=rand(n, n)，其中n=2, 4, 8, 16, 32, ...128。将invByCol函数的浮点操作次数与内置命令inv的进行比较。注意数量级估计只有在n足够大时才适用。MATLAB第六版的用户不能用flops函数来测量inv函数的浮点操作次数。如果是那样的话，矩阵求逆的运算量估计值的数量级为$O(n^3)$。
22. (2)编写一个parabola函数，自动地建立由$y = c_1x^2 + c_2x + c_3$决定的抛物线的方程组并解此方程组。此函数的定义应该为：

```
function c = parabola (x, y)
```

此函数有两个输入向量x和y，其长度都为3，定义了通过抛物线的三个点。此函数应该返回含三个系数c_i的向量。

分别求出通过下列点集的抛物线方程。对每一个点集，使用c_i来产生抛物线上的100个点的向量，并沿原始的三个点画出这些点，验证得到的抛物线方程是正确的。

(a) (−2, −1), (0, 1), (2, 2)
(b) (−2, −1), (1, −1), (2, 0)
(c) (−2, −2), (−1, 1), (2, −1)

(d) (−2, 2)、(0, 0)、(2, −2)

(e) (−2, 2)、(−1, −2)、(−1, 2)

23. (2) 考虑平面方程$a_1x+a_2y+a_3=z$。已知三个点(x_1, y_1, z_1)、(x_2, y_2, z_2)、(x_3, y_3, z_3)，写出用决定a_1、a_2和a_3的方程组。编写一个plane函数，当给定三个点时能求解出相应的a_i。求解通过(1, 0, 0)、(0, 1, 0)和(0, 0, 1)三个点的平面方程，以此验证你的函数。在(x, y) = (0, 0.5)、(0.5, 0)、(0.25, 0.25)和(0.5, 0.5)处，z的值是多少？

24. (2)若$R_1=R_3=R_4=R_5=10\text{k}\Omega$、$R_2=20\text{k}\Omega$，那么例8.5中的3×3系数矩阵的秩是多少？在$V_{in}=5\text{V}$下，此系数矩阵相容吗？在相同的阻值下，关于$v_b$和$v_c$的2×2系数矩阵的秩是多少？已知$V_{in}=5\text{V}$，那么$v_a$、$v_b$、$v_c$和$v_d$的值是多少？

449

25. * (2+)求解例8.5中的3×3系数矩阵中的奇异点还有另一种方法，就是修改矩阵中的元素。编写一个关于v_b、v_c和v_d的平凡(trivial)方程，其解$v_d=0$。使用此方程替换例8.5中3×3方程组中关于v_d方程。假设$R_1=R_3=R_4=R_5=10\text{k}\Omega$、$R_2=20\text{k}\Omega$且$V_{in}=5\text{V}$，解得到的3×3方程组，求出$V_{out}$的值。

26. (2)在区间$1\text{k}\Omega \leqslant R_2 \leqslant 100\text{k}\Omega$中，画出例8.5中的惠斯通电桥的$V_{out}=v_b-v_c$与$R_2$的关系图。假设$R_1=R_3=R_4=R_5=10\text{k}\Omega$且$V_{in}=5\text{V}$。将系数矩阵作一次分解，然后再重用它，这样是否可以减少计算量呢？（提示：编写一个函数，将电阻值作为一个向量输入，自动建立和求解方程组以得出解来。）

27. (2)假设$R_1=R_3=R_4=R_5=10\text{k}\Omega$，$R_2=20\text{k}\Omega$且$V_{in}=5\text{V}$，解方程（8-19）。$V_{out}=v_b-v_c$的值是多少？

28. (2)一个油漆公司将一些不受欢迎的油漆颜色加以混合，得到受欢迎的油漆颜色。每种油漆都含四种基本的色素。下表列出了四种不受欢迎的油漆颜色的组成：

油漆编号	色素成分			
	A	B	C	D
1	80	0	16	4
2	0	80	20	0
3	30	10	60	0
4	10	10	72	8

期望得到的颜色是40%的色素A、27%的色素B、31%的色素C和2%的色素D。要得到一加仑这种颜色的油漆，需要表中四种颜料各多少？（提示：令x_j为配制这种油漆所需要的一加仑不受欢迎的油漆j的百分比。那么，新油漆中色素A的总量为$80x_1+30x_3+10x_4=40$。）

29. (2+)下图描述了包含电阻和电源的电路。根据Kirchhoff电压定律，电路中任意一个回路的电压降的和为零，可以得到如下方程：

$$
\begin{aligned}
i_1R_1+v_1+(i_1-i_2)R_2-v_3&=0\\
(i_2-i_1)R_2+i_2R_3+v_2+i_2R_4+(i_2-i_4)R_5&=0\\
i_3R_6+v_3+(i_3-i_4)R_8+v_4+i_3R_7&=0\\
-v_4+(i_4-i_3)R_8+(i_4-i_2)R_5+i_4R_9+v_5&=0
\end{aligned}
$$

其中i_k是回路k中的电流。

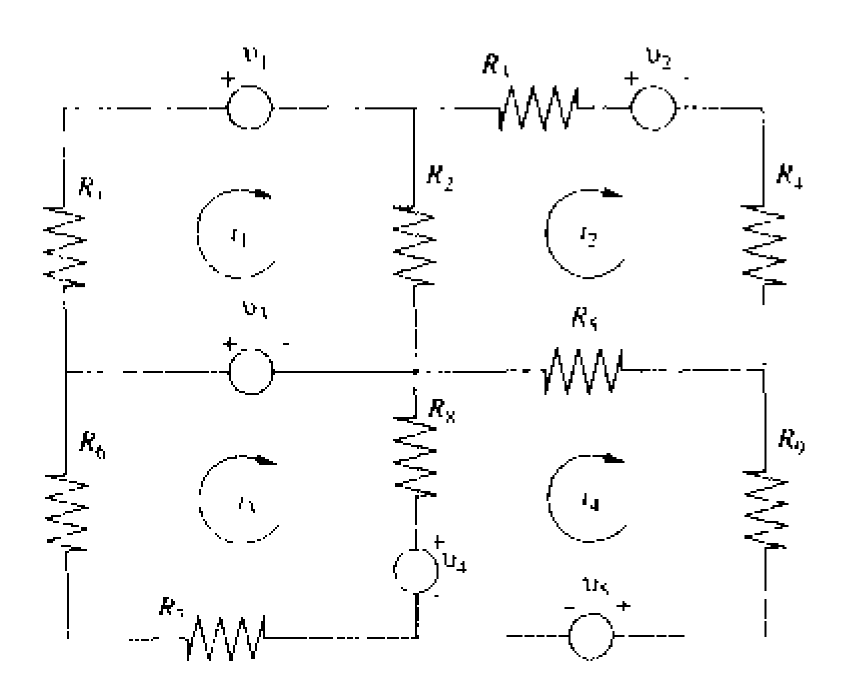
(a) 将这些方程写成$\boldsymbol{R}i=v$的形式，其中$\boldsymbol{R}$是一个矩阵，i和v是向量。

(b) 已知下表中电阻的阻值和电源的电压（降），计算通过每个回路的电流。

450

电阻	1	2	3	4	5	6	7	8	9
电阻值(kΩ)	1	3.5	4	1.75	1	1	2	1.5	5

电源编号	1	2	3	4	5
电源电压v(V)	5	5	2.5	2	5



30. (2)对任意的对角矩阵

$$
\boldsymbol{D}=\operatorname{diag}(d_1,d_2,\cdots,d_n)=\begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}
$$

求出条件数$\kappa_1(\boldsymbol{D})$的表达式。证明：如果对一些标量α,d_i都相等（即$\boldsymbol{D}=\alpha\boldsymbol{I}$），那么$\kappa_1(\boldsymbol{D})=\alpha^2$，并推断出$\kappa_1(\boldsymbol{I})=1$。

31. (1+)证明对任意$\boldsymbol{A}$有$\kappa(x)\geqslant 1$。（提示：由$\|\boldsymbol{I}\|=\|\boldsymbol{A}\boldsymbol{A}^{-1}\|$和上题中的最后结果开始计算。）

32. (2)修改例8.1中的`pumpCurve`函数，使之能够接受任意长度的输入向量q和h。既然插值多项式的次数由`length (q)-1`决定，那么它就不需要明确地计算出来（编程提示：`h = h(:)`、`help vander`）。你所修改后的`pumpCurve`函数除了要计算多项式的系数，还要计算矩阵$\boldsymbol{A}$的条件数。由下表中的数值求出3次和4次插值多项式的系数。

$q(\mathrm{m^3/s})$	0.0001	0.00025	0.0008	0.001	0.0014
$h(\mathrm{m})$	115	114.2	110	105.5	92.5

使用表中的第一、第二、第三和第五个点来定义3次多项插值式。对两个插值式求出其多项式系数和$\boldsymbol{A}$的条件数。条件数随着多项式的次数如何改变？

451

33. (3)使用练习32编写的`pumpCurve`函数研究输入数据扰动的影响。特别地，将第二个h的值$h=114.2$代换为$h=114$，重新计算三次插值多项式的系数。令$\tilde{c}$为扰动数据所得三次插值多项式的系数、c为原始数据所得多项式的系数。对每个多项式系数，相对差$(\tilde{c}_i-c_i)/c_i$是多少？在区间$\min(q)\leqslant q\leqslant\max(q)$的100个数据点上对两个三次插值多项式计算$h(q)$并画图。由扰动数据和原始数据所得到的插值式的$h$的最大差是多少？通过扰动$c$值导致插值式所得结果$h$的变化来讨论扰动对实际问题影响的重要性。

34. (3)将q数据转化成$\mathrm{m^3/hour}$，重复习题33中的计算。其中哪个结果改变了？哪个结果

在比例上没有影响？本问题中缩放输入数据的比例有什么好处？

35. (2)编写一个condSurvey函数，计算由内置rand函数产生矩阵的$\kappa_2(A)$并画图。在一个维数逐渐增加的序列（比如n = [4 8 16 ...128]）上，产生10个随机的$n \times n$矩阵（每个n作10个）并计算其条件数。作出变量$\kappa_2(A)$和n的对数-对数图，观察其趋势。

36. (2)令x为$Ax=b$的精确解，$\hat{x}$ 为其数值解。定义残差为$r = b - A\hat{x}$，要求推导出方程(8-37)（提示：证明$r = A(x - \hat{x})$并利用方程（8-28））。

37. (2)在例8.9中，使用MATLAB的反斜杠运算符求来解形式为

$$\begin{bmatrix} 1 & 1 \\ 2 & (2+\delta) \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 \\ 2 & (2+\delta) \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1.001 \\ 2 \end{bmatrix}$$

的两个方程组。利用高斯消去法手动计算求出这两个方程组的精确解。当$\delta=100\varepsilon_m$时，反斜杠运算符能返回正确结果吗？当$\delta=\varepsilon_m/100$时，反斜杠运算符能返回正确结果吗？方程组对b的扰动敏感取决于在计算中使用单精度还是双精度？

38. (1)证明方程（8-46）中矩阵的逆为

$$M_{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ -m_{21} & 1 & 0 \\ -m_{31} & 0 & 1 \end{bmatrix} \quad \text{和} \quad M_{(2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -m_{32} & 1 \end{bmatrix}$$

（提示：乘积等于I吗？）

452

39. (2+)由程序清单8-1中的GEshow函数，编写一个GEmultiplier函数，当输入矩阵A时，返回矩阵M，使得MA为三角矩阵。此函数应该能够在

$$A = \begin{bmatrix} -3 & 2 & 1 \\ 6 & -6 & 7 \\ 3 & -4 & -4 \end{bmatrix} \quad \text{时，有} \quad MA = \begin{bmatrix} -3 & 2 & -1 \\ 0 & -2 & 5 \\ 0 & 0 & -2 \end{bmatrix}$$

求出满足上面对矩阵A重新排列的矩阵M。

40. (3+)开发一个m文件函数，使用A的原位LU分解来求解$Ax = b$。特别地，

(a) 修改luPiv，将修改后的函数称为luInPlace，它返回A的L、U因子并存储在A所占用的地址空间中。函数luInPlace的函数定义行（第一行）应该为

```
function [A, pv] = luInPlace(A, ptol)
```

(b) 编写一个称为luSolveInPlace的函数，将A和b作为输入，解出$Ax = b$并返回x。此函数使用luInPlace返回的分解因子，不明确地计算L和U，解出这两个三角方程组。利用习题16和习题117中的结果，可以减少求解的步骤。注意：需要修改习题16中产生的lsolve函数，以便乘以L(i,i)或者除以L(i,i)的操作可以用包含的隐含计算式来代替。

(c) 通过比较此函数的运算结果和 \ 运算符产生的结果，来验证luSolveInPlace函数的正确性。

41. (2+)下列MATLAB语句明显地不相容，解释其原因。

```
>> L = [1 0 0; 2 1 0; 3 1 1];
>> U = [2 0 1; 0 2 1; 0 0 2];
>> [L2,U2] = lu(L*U)
```

这是否意味着LU分解不是惟一的?

42. (3+)已知Cholesky分解$A=U^tU$，求相关的对称分解$A=LDL^t$，其中L是对角线元素为1的下三角矩阵，D是一个对角矩阵。编写一个LDLT函数，返回矩阵L和包含D的对角线元素的向量d。使用chol来计算LDLT函数内的Cholesky分解，分解一个对角线上是2、对角线上下都是-1的$n\times n$三对角矩阵，以验证你的函数。将此矩阵作为A，且$b=[1, 0, \ldots, 0]^t$，解$Ax=b$（提示：$A=U^tU=LDL^t=(LD^{1/2})(D^{1/2}L^t)$，其中$D^{1/2}$是对角矩阵且$(D^{1/2})(D^{1/2})=D$）。另外，注意$L^t=(D^{1/2})U$只是$U$的行的缩放。

43. (3)修改上题中的LDLT函数，使任何行（或列）的缩放操作都避免与0相乘，以将工作量降低到最小。比较这两种方法的浮点操作次数。

44. (2+)修改例8.10方程组中的A和b，使A与x无关，而b与x相关（提示：将所有的非线性项移到右边）。修改demoSsub，使用新的A和b解方程组。使用修改的函数进行15次逐次代入迭代操作，设初始猜测值$x^{(0)}=[1, 1]^t$，$[2, 2]^t$，…$[5, 5]^t$。

453

45. (3-)修改习题44中开发的m文件函数，使用lu函数来分解所求解方程组的系数矩阵。既然A是常数，就只在主循环外进行一次LU分解。在主循环内使用内置的\运算符来进行三角方程组求解。使用flops函数来测量作10次迭代后LU分解所节省的计算量。如果A是$n\times n$矩阵，用数量级来表示出k次迭代之后此方法所节省的浮点操作次数。

46. (2)修改demoNewtonSys函数，使其能够使用newtonSys函数来求解例8.13中的非线性方程组。

47. (4)对于习题29中的电路，求出使$i_4=-0.85\text{mA}$的R_5的值。解此问题有不同的方法，但归根结底是一个求根过程：

- 手动迭代直到i_4“接近”-0.85mA。
- 用手动迭代产生的数据画图，得到图形解。
- 把回路方程的解嵌入到求根程序中。期望求根的函数是$f(R_5)=0$。对$f(R_5)$的每一步计算都需要在R_5的猜测值上求解一次线性方程组。返回值是由R_5的猜测值计算出的i_4值和期望的i_4值之间的差。
- 使用牛顿法解方程组。

454

第9章 数据的最小二乘曲线拟合

曲线拟合是寻找一个相对简单的解析函数来逼近（近似）一个数据集的过程。此数据集的数据通常来源于实验测量，可能包含错误数据，或者数据的有效数字的位数有限。为此要调整逼近函数的参数，使其计算数据与实验测得数据相一致。通常情况下，拟合的性能通过计算近似函数值和给定数据集中数值的差的平方和来度量。根据最小二乘（法）(*least square*)，平方和最小的拟合最好。

线性的最小二乘曲线拟合问题涉及一个基本函数集$f_j(x)$，$j=1,\ldots,n$，以及未知的拟合系数c_j，这二者决定了一个拟合函数，形式为：

$$F(x)=c_1f_1(x)+c_2f_2(x)+\ldots+c_nf_n(x) \tag{9-1}$$

对一个给定的数据对集合(x_i,y_i)，$i=1,\ldots m$，根据最小二乘法原理可以提供一种方法确定系数c_j，使$F(x_i)\approx y_i$。一般地，数据对的个数m比未知系数的个数n要多，因此找不到c_j使$F(x_i)=y_i$严格成立。

例如，如果$f_1(x)=x$，而$f_2(x)=1$(常数)，那么，

455

$$F(x)=c_1x+c_2 \tag{9-2}$$

这时可用一条直线来拟合数据。线性最小二乘问题就是要求解出函数$F(x)$，它线性依赖于c_j，如方程（9-1）所示。$F(x)$并不要求一定是个线性函数。例如，若$f_1(x)=x^2$，$f_2(x)=x$，$f_3(x)=1$，那么逼近函数$F(x)$就是一个二阶多项式。

在统计学文献中，曲线拟合叫做“回归”(regression)。完全的回归分析需要解最小二乘问题，然后检查输入数据的统计属性和拟合函数。除了如何求最小二乘，回归也涉及到拟合程序。本章着重表述的是最小二乘问题的数值限定与求解方面，而不是对数据进行统计分析。用户应该能轻易对文中提供的m文件进行扩展，使其包含统计分析。本章的“补充读物”部分概要介绍了回归的统计分析方面的相关参考读物。

图9-1是本章内容的概述。开始我们先推导数据的直线拟合公式，如方程（9-2）。斜率和截距的最小二乘解可通过m文件函数实现，然后将直线拟合程序扩展成为函数，使其能够通过数学变换变成线性关系。变换的一个典型例子是指数式衰减（exponential decay）数据的拟合，这种情况下，输出数据(y)的对数线性依赖于输入数据(x)。

本章主题

1 数据的直线拟合

本节推导出了直线的最小二乘拟合公式，为此构造了一个m文件，在得出正规方程组（normal equtions）的解的基础上进行拟合，这也证明了可以使用线性变换将一些简单的非线性拟合问题转化为线性拟合问题

2. 对线性组合函数的数据拟合

这里阐明了方程$y=F(x)$的最小二乘法数据拟合问题，其中$F(x)$是函数的一个线性组合，为此构造了相应的m文件，解出正规方程组并用QR（因式）分解（QR factorization）得到超定方程组（overdetermined system）的解，以对数据进行拟合。同时还介绍了内置的polyfit函数对多项式的拟合。

3. 多元最小二乘拟合

这里将线性最小二乘问题加以扩展，使因变量y是包含p个独立参数的函数（即$y=F(x_1,\ldots,x_p)$）。

456

图9-1 第9章的主题

最小二乘直线拟合程序构造后，再推导对任意基本函数的线性组合进行拟合的一般情况。最小二乘的拟合系数可以通过解正规方程组得到，也可用QR分解法直接“解”某个超定方程组而得到。这两种方法在本章中都进行了推导。另外本章还给出了对应的m文件，以实现任意基本函数组合的曲线拟合。用这些m文件解决的问题中，数据的数学模型一般都事先知道，若数据的数学模型未知，那么一般就用一个多项式来作拟合函数。这时，使用内置函数polyfit就可以很方便地用多项式进行曲线拟合。文中会通过一个例子来说明polyfit函数的用法。

本章的第三节阐述了更为一般化的多元最小二乘拟合问题，这种情况下，因变量y变成了多个自变量x_1，x_2，…，x_n的函数。

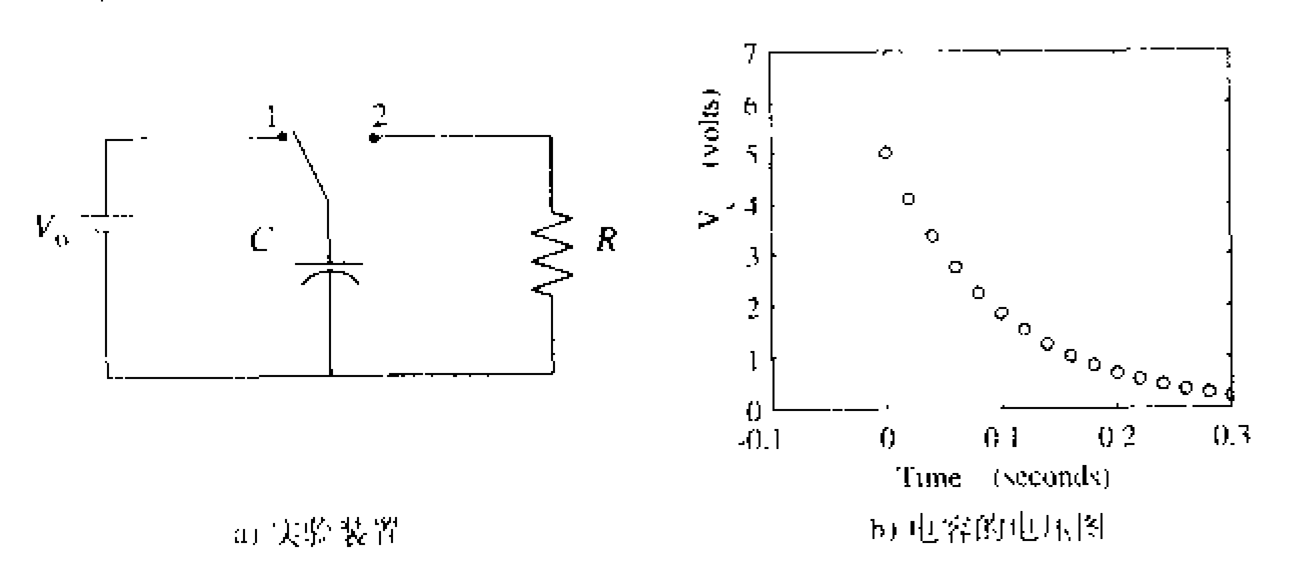
例9.1　电容放电的时间常数

图9-2a是一个在电容放电时测量电压降的简单实验。开关一开始放在位置1，电流对电容充电。在$t=0$时刻，将开关扳到位置2处，这样电流就流过电阻，电容放电。电容电压随时间的变化如图9-2b所示。

电容的电压与时间的函数关系预计如下：

$$V = V_0 \exp\left(-\frac{t}{RC}\right)$$

如果已知$V_c(t)$数据，就可以用最小二乘拟合来求时间常数$\tau = 1/(RC)$。例9.5给出了此曲线拟合在MATLAB中的实现。



a) 实验装置　　b) 电容的电压图

图9-2　在电容放电过程中测量电压

457

例9.2　低温条件下铜的导热性

开发高速电路的工程师们一直想利用极低温度条件下材料的超导性。温度降低后，半导体材料的电阻也随之减小，电路的时钟速度就可以增大。此技术应用的挑战之一，就是如何将电信号输入输出降温后的实验电路。

通过铜导线传递的热量是超导电路负载热量中很重要的一部分。为了对导线中的热流精确建模，必须要考虑到铜的导热性在不同温度下的变化情况。图9-3描述了三种铜导线的导热性测量值。由其中的数据可以看出，热传导率K到达一个最大值后，在温度接近绝对零度时降低到零。有了$K(T)$的曲线拟合，这些实验数据就可以在热学设计的计算中使用。图9-3中所使用的导热性数据保存在NMM工具箱的data目录下的cucon1.dat、cucon2.dat和cucon3.dat文件中。图9-3的实线由9.2.2节的例9.9中的最小二乘曲线拟合得到。

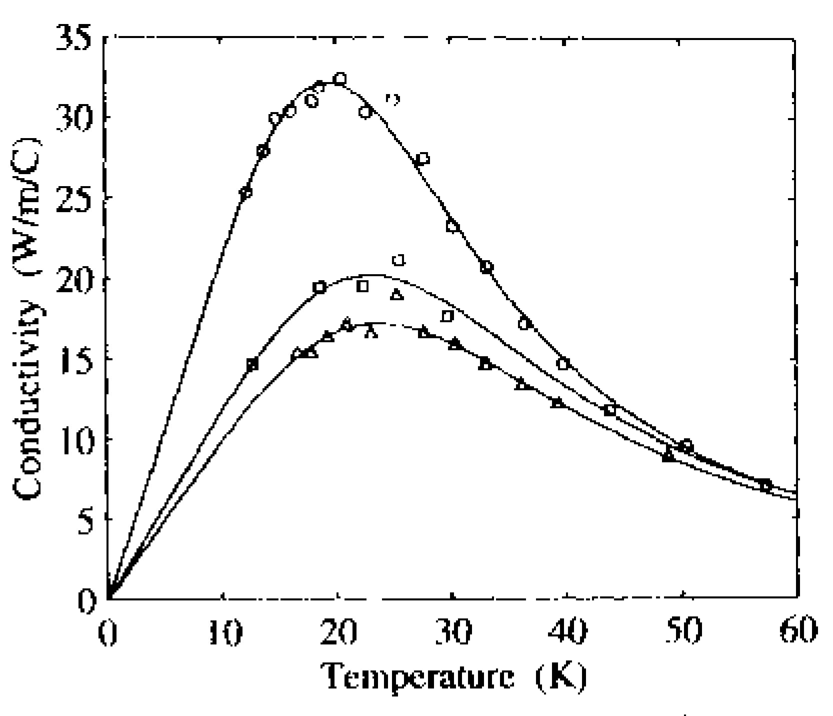


图9-3　低温条件下铜导热性的测量值

9.1　数据的直线拟合

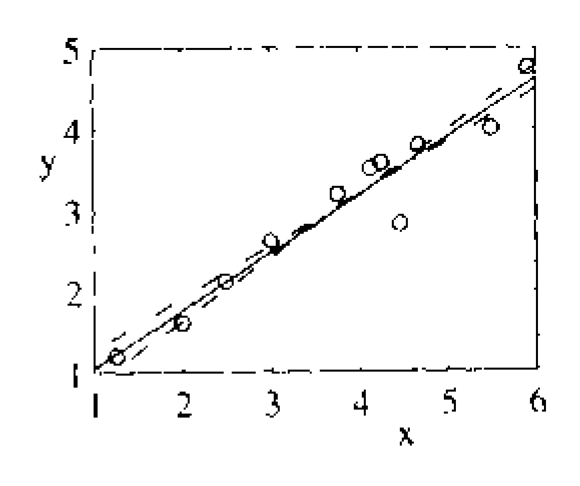
最简单的最小二乘拟合问题是求一条直线的斜率和截距，这条直线是对一个数据集的逼近，这种问题在哪儿都可以解决，许多掌上计算器就有处理这种问题的内置程序。求斜率和截距的计算非常简单，有的读者可能已经略过这里直接去看9.1.3节中介绍的linefit函数。虽然直线拟合的实现很简单，但是关于生成拟合系数的方程推导还是很有用处的，同样的基本过程和基本原则可应用于函数的任意线性组合问题，这些内容更有趣。

[458]

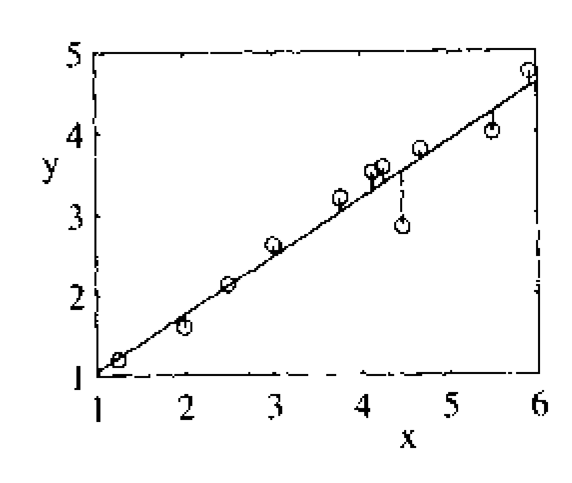
图9-4表示一个数据集可以用一条直线来逼近

$$y = F(x) = \alpha x + \beta \tag{9-3}$$

图9-4a中，从斜率和截距稍微不同的直线中可以选出最好的数据拟合。通过调整α和β，能够降低一些点的拟合效果，而改进另外一些点的拟合。一般不可能找到一条直线，能同时很好地逼近所有的数据点。从图9-4可以看出，这些直线似乎能拟合大部分点，但有一个$x \approx 4.5$的点明显地在它们的下面。因此，优化地选取α和β，可以使直线只拟合大部分点的趋势，而不去管那些远距离的游离点。然而，α和β的选择程序应该让所有数据点的拟合质量有一定程度的平衡。



a) 一些数据的曲线拟合的候选直线x=4.5附近有一个点是个游离点



b) 最小二乘拟合使距离$y_i - F(x_i)$的平方和最小，数据与图9-4a的相同

图9-4　直线的最小二乘拟合

假设我们测出拟合函数与每个数据点的距离，然后将这些距离综合，得到一个拟合性能的总体评估。设r_i为每个数据点到直线的纵向距离，即：

$$r_i = y_i - F(x_i) = y_i - (\alpha x_i + \beta) \tag{9-4}$$

选择符号r是因为r_i是描述拟合的方程组的残差（参照9.1.2节），是数据点和拟合函数之间的纵向距离，如图9-4b。既然r_i可正可负，我们就不能简单地将他们相加来对拟合作总体评估。这里可以使用两种方法综合r_i [459]

$$\sum |r_i| \quad 和 \quad \sum r_i^2$$

使用$\sum r_i^2$来评估拟合质量有两个明显的好处。首先，可以用较简单的程序计算α和β（或更一般地，计算方程（9-1）中的c_j）。其次，如果适当约束数据(x_i, y_i)，那么使$\sum r_i^2$最小的α和β值就很可能（在统计意义上）是给定(x_i, y_i)的α和β值（例见[19]）。

设ρ为r_i的平方和，即

$$\rho = \sum_{i=1}^{m} r_i^2 = \sum_{i=1}^{m} [y_i - (\alpha x_i + \beta)]^2 \tag{9-5}$$

给定数据(x_i, y_i)，则方程（9-5）中就只有ρ、α和β（都为标量）未知。以α和β表示ρ，并将其最小化，就决定了一条直线，实现数据对直线的最小二乘拟合。其中“最小”表示最小化过程，“二乘”表示最小化的数量，即$\rho = \sum r_i^2$。运用最小二乘法可求解出一个直线方程，它对应于方程（9-5）中所定义函数$\rho(\alpha, \beta)$的最小值。在下节中，将使用两种方法来推导求解α和β的方程

9.1.1 求残差最小值

我们的直接目标是推导出一个计算过程，求解出使$\rho=\rho(\alpha, \beta)$最小的α和β。最小值的必要条件是

$$\left.\frac{\partial \rho}{\partial \alpha}\right|_{\beta = \text{constant}} = 0 \quad 和 \quad \left.\frac{\partial \rho}{\partial \beta}\right|_{\alpha = \text{constant}} = 0 \tag{9-6}$$

展开微分式得

$$\frac{\partial \rho}{\partial \alpha} = \sum_{i=1}^{m} \frac{\partial}{\partial \alpha}[y_i - (\alpha x_i + \beta)]^2 = \sum_{i=1}^{m} 2(-x_i)[y_i - (\alpha x_i + \beta)]$$

$$\frac{\partial \rho}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial}{\partial \beta}[y_i - (\alpha x_i + \beta)]^2 = \sum_{i=1}^{m} 2(-1)[y_i - (\alpha x_i + \beta)]$$

[460]

故，要使条件(9-6)成立，要求

$$0 = \sum_{i=1}^{m} x_i y_i - \alpha \sum_{i=1}^{m} x_i^2 - \beta \sum_{i=1}^{m} x_i$$

$$0 = \sum_{i=1}^{m} y_i - \alpha \sum_{i=1}^{m} x_i - \beta m$$

移项，得

$$\left(\sum x_i^2\right)\alpha + \left(\sum x_i\right)\beta = \sum x_i y_i \tag{9-7}$$

$$\left(\sum x_i\right)\alpha + m\beta = \sum y_i \tag{9-8}$$

为简便，去掉了其中求和范围的限制。方程（9-7）和方程（9-8）是数据对直线的最小二乘拟合的正规方程组。手工解此关于α和β的方程组，得

$$\alpha = \frac{\left(\sum x_i\right)\left(\sum y_i\right) - m\sum x_i y_i}{\left(\sum x_i\right)^2 - m\sum x_i^2}, \quad \beta = \frac{\left(\sum x_i\right)\left(\sum x_i y_i\right) - \left(\sum x_i^2\right)\left(\sum y_i\right)}{\left(\sum x_i\right)^2 - m\sum x_i^2} \tag{9-9}$$

方程（9-7）和方程（9-8）也可以写成

$$\begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & m \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix} \tag{9-10}$$

在建立一个m文件来解方程（9-10）之前，我们用另一种方法来重新推导这个方程。

9.1.2 超定方程组

前面的推导使用几何方法求最小二乘拟合：选择α和β的值使拟合函数和数据点在y方向上的差别最小。另外，曲线拟合也可以直接表示为m个关于α和β的联立线性方程组。这里未知量个数少于方程个数，不太可能得到精确解，要得到解需要添加一些限制条件。

试想一下将三个数据对(x_1, y_1)、(x_2, y_2)和(x_3, y_3)拟合为一条直线的情况。当然，对穿过任意两个数据对的直线，都能够很简单地求解出其斜率和截距。例如，对于过第一个和最后一个数据对的直线，有α和β使

461

$$\begin{aligned} \alpha x_1 + \beta &= y_1 \\ \alpha x_3 + \beta &= y_3 \end{aligned} \quad 或者 \quad \begin{bmatrix} x_1 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_3 \end{bmatrix}$$

成立。这条直线虽然完全精确地穿过第一个和最后一个数据对，但却不能使方程（9-5）中定义的ρ最小。

但是为什么选择集合中的第一个和最后一个数据对呢？或许，我们可以写出穿过每个点的直线的方程，这样得到一个非方阵系数矩阵的方程组。

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{9-11}$$

左边两个矩阵的内部维数是匹配的，故此方程符合线性代数的规则(参照7.2.2节)。

前面的推导可以推广到包含m个数据对(x_i, y_i)的集合的情况，$i = 1,\ldots,m$。如果我们将y_i表示成关于x_i的m个线性方程，得：

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

此方程可简写为

$$\boldsymbol{Ac} = \boldsymbol{y} \tag{9-12}$$

其中，

$$A=\begin{bmatrix}x_1 & 1\\ x_2 & 1\\ \vdots & \vdots\\ x_m & 1\end{bmatrix},\quad c=\begin{bmatrix}\alpha\\ \beta\end{bmatrix},\quad y=\begin{bmatrix}y_1\\ y_2\\ \vdots\\ y_m\end{bmatrix} \tag{9-13}$$

只有当y沿着由某对α和β决定的直线分布时，方程（9-12）才有精确解。如8.1.2节所述，这表示只有y处于矩阵A的列空间时，方程才有精确解。既然不太可能得到精确解，则残差向量

$$r=y-Ac \tag{9-14}$$

就不会为零。最小二乘法提供了一个使$\|r\|_2$最小的折衷“解”。直接求方程（9-14）中的$\rho=\|r\|_2^2$的值，得：

462

$$\begin{aligned}\rho=\|r\|_2^2=r^Tr&=(y-Ac)^T(y-Ac)\\&=y^Ty-y^TAc-c^TA^Ty+c^TA^TAc\\&=y^Ty-2y^TAc+c^TA^TAc\end{aligned}$$

将关于向量c的ρ最小化，要求$\partial\rho/\partial c=0$。使用线性代数规则求其导数得

$$\frac{\partial\rho}{\partial c}=-2A^Ty+2A^TAc$$

为使ρ的最小值存在，必有　$A^Ty+A^TAc=0$，或

$$\boxed{(A^TA)c=A^Ty} \tag{9-15}$$

方程（9-15）是最小二乘曲线拟合正规方程组的一般形式。直接计算可知方程（9-15）与方程（9-10）在直线拟合问题上是等价的（参照练习5）。

9.1.3 直线拟合的实现

前面讨论了数据的直线最小二乘拟合的不同见解，现在我们开始编制m文件来实现这些方法。我们要在方程（9-13）中定义矩阵A和向量y，然后在方程（9-15）中解出向量c。另外，直线的斜率和截距也可从方程（9-9）中解出（参照习题4）。这里我们采用前一种方法，由于它为一个更一般的问题提供了例子，这个问题就是以后讨论的任意函数组合的最小二乘拟合问题。

程序清单9-1中的`linefit`函数用来构造矩阵A，并解出方程（9-15）。此函数有下面两种调用形式：

```
c = linefit(x,y)
[c,R2] = linefit(x,y)
```

输入向量x、y的长度相同，而输出c则是只包含了两个元素的向量，这两个元素就是最小二乘直线拟合的斜率和截距。可选的第二个参数R2表示R^2统计量（将会在9.1.4节中讨论）。例9.3示范了如何使用`linefit`函数。

例9.3　四个数据点的直线拟合

设想下四个数据点(1,1)、(2,2)、(4,2)和(5,3)对一条直线的拟合。一旦指定了数据(x,y)，剩下的工作就可以由`linefit`函数来完成。

463

程序清单9-1　linefit函数对数据对(x, y)进行最小二乘直线拟合，并返回直线的斜率和截距

```
function [c,R2] = linefit(x,y)
% linefit   Least-squares fit of data to y = c(1)*x + c(2)
%
% Synopsis:  c      = linefit(x,y)
%            [c,R2] = linefit(x,y)
%
% Input:   x,y = vectors of independent and dependent variables
%
% Output:  c  = vector of slope, c(1), and intercept, c(2) of least sq. line fit
%          R2 = (optional) coefficient of determination; 0 <= R2 <= 1
%               R2 close to 1 indicates a strong relationship between y and x
if length(y)~= length(x),  error('x and y are not compatible');  end

x = x(:);  y = y(:);      %  Make sure that x and y are column vectors
A = [x ones(size(x))];    %  m-by-n matrix of overdetermined system
c = (A'*A)\(A'*y);        %  Solve normal equations
if nargout>1
  r = y - A*c;
  R2 = 1 - (norm(r)/norm(y-mean(y)))^2;
end
```

```
>> x = [1 2 4 5];    y = [1 2 2 3];
>> c = linefit(x,y)
c =
    0.4000
    0.8000
```

通过绘制原始数据点图和拟合方程曲线图，可以定性地检查拟合成功与否。下面的语句将产生图9-5中的图形。

```
>> xfit = [0 6];                  %  Evaluate fit over this range of x
>> yfit = c(1)*xfit + c(2);       %  Values of the fit function
>> plot(x,y,'o',xfit,yfit,'-')
>> grid on;
>> xlabel('x values'),
>> ylabel('y data and fit function');
```

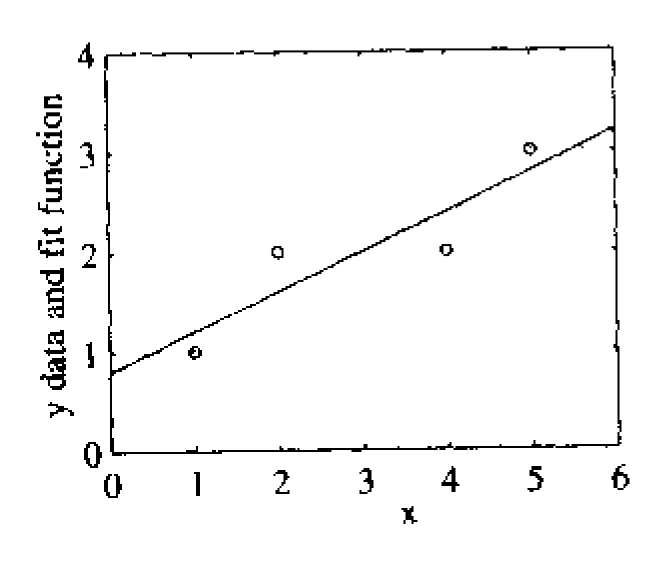


图9-5　四个数据点的直线最小二乘拟合

9.1.4　R^2统计量

464

通过观察拟合图，可以看出给定数据的曲线拟合质量。本节中我们将讨论拟合质量的一

个定量化指标。

令 $\hat{y}$ 为已知数据点处拟合方程的y值，即

$$\hat{y}_i = c_1 x_i + c_2 \tag{9-16}$$

由方程（9-4）可知，$r_i = y_i - \hat{y}_i$，另外 $\|r\|_2^2 = \sum_{i=1}^{m}(y_i - \hat{y}_i)^2$。

现在，我们不用直线来逼近数据，而是设想一个更简单的模型，由数据的平均值来作逼近，换言之，就是用$F(x) = \bar{y}$ 来逼近数据，其中：

$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

这个简单模型是否比直线拟合更有用呢？此问题可更明确地描述为：增加第二个参数即直线斜率对数据拟合的改进能有多大的作用？为了回答这个问题，我们定义一个决定系数（*coefficient of determination*）：

$$R^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2} \tag{9-17}$$

上式中的两个求和运算的范围都为$i = 1,\ldots,m$。易知

$$\sum(y_i - \bar{y})^2 = \sum(y_i - \hat{y}_i)^2 + \sum(\hat{y}_i - \bar{y})^2$$

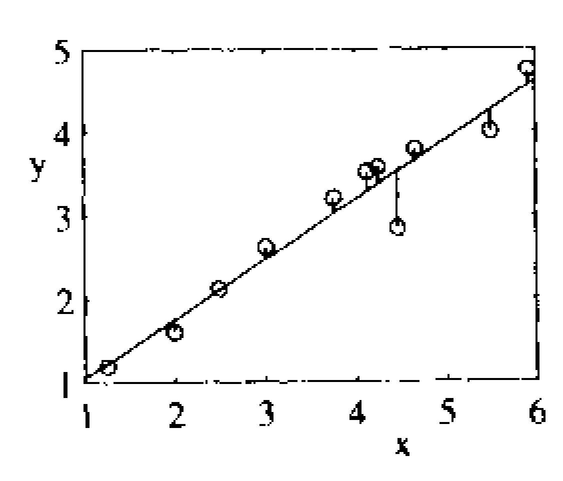
将此结果代入方程（9-17），得：

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} = 1 - \frac{\|r\|_2^2}{\sum(y_i - \bar{y})^2} \tag{9-18}$$

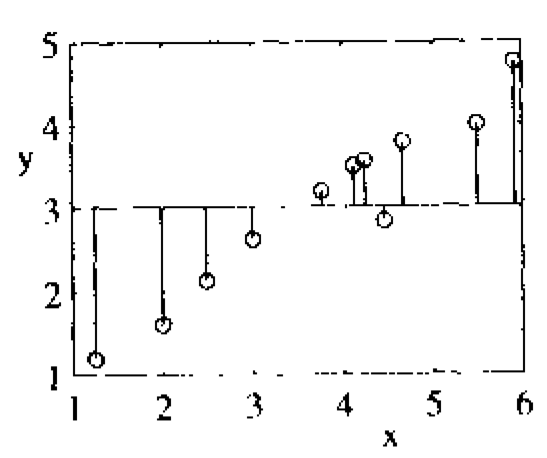
465

这个R^2表达式很清楚地表达了拟合视在质量（apparent quality）与残差的关系。最小二乘法能使 $\|r\|_2$ 最小，但只要拟合函数不能通过每个(x_i, y_i) 点，就有 $\|r\|_2 \neq 0$。如果残差的平方与 $\sum(y_i - \bar{y})^2$ 相比较很小，则拟合曲线会距离数据点很近，R^2的值也接近1。

图9-6a表示一个数据集的直线拟合，每个数据点的残差与空心圆和最小二乘直线之间的纵向距离成比例。图9-6b表示相同的数据，另外还有一条经 $\bar{y}$ 的水平直线，$y_i - \bar{y}$ 的值与图9-6b中的垂线长度成比例。在这些数据的拟合直线中，有$R^2 = 0.934$，说明方程（9-18）中 $\|r\|_2 \ll \sum(y_i - \bar{y})^2$，这与图9-6的视觉印象一致：图9-6a中最小二乘拟合直线的拟合效果明显好于图9-6b中的通过 $\bar{y}$ 的水平直线的拟合效果。



a) 用来计算 $\|r\|_2^2 = \sum(y_i - \hat{y}_i)^2$ 的距离



b) 用来计算 $\sum(y_i - \bar{y})^2$ 的距离，其中水平线经过 $\bar{y}$

图9-6　计算R^2的数据的图形化描述

方程（9-17）中的分子表示拟合函数与数据y的平均值的差别程度。若此数与分母相比很小，就可以用一个常数(即 $\bar{y}$ ）来逼近数据。方程（9-17）中的分母表示数据与其平均值的差别程度，既然拟合函数最终会将这些差值平衡（平均化），那么方程（9-17）中的分子就不会比分母大。另外，由于分母和分子都是正数，有$0 \leq R^2 \leq 1$。

R^2的值可作为拟合质量的指标。若它接近0，表示y和x之间没有很密切的关系，反之若它接近1，则说明y和x之间的关系很密切。但较大的R^2并不能保证这种视在关系（apparent 466 relationship）就一定成立（参照练习13）。

在9.2.2节中将再次提到R^2，到时将与读者一起讨论调整过的决定系数（*adjusted coefficient of determination*）。R^2将作为`linefit`函数的第二个参数（可选）返回。

例9.4 金刚砂的体积模量与温度的关系

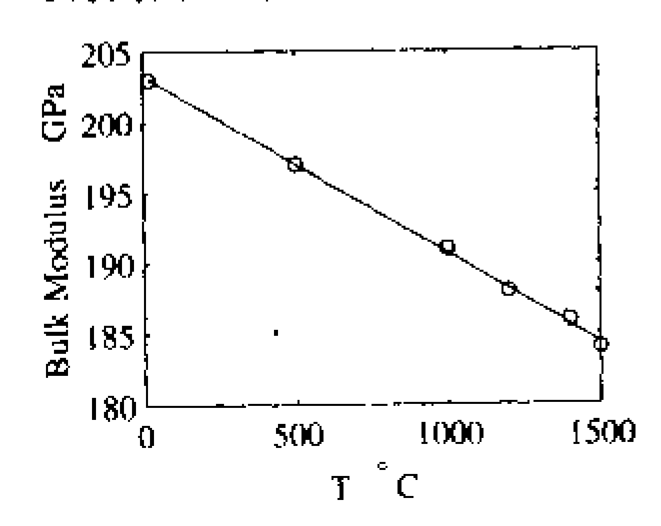
R.G.Munro[⊖]发布了一种烧结金刚砂（Silicon Carbide，SiC）的材料属性数据。NMM工具箱中`data`目录下的`SiC.dat`文件摘录了其中一段。金刚砂的体积模量随温度的变化情况如下表

T(°C)	20	500	1000	1200	1400	1500
G(G Pa)	203	197	191	188	186	184

下面的MATLAB语句用来加载`SiC.dat`文件中的数据，并可得到金刚砂的体积模量-温度函数的最小二乘直线拟合。

```
>> [t,D,labels] = loadColData('SiC.dat',6,5);
>> g = D(:,1);
>> [c,R2] = linefit(t,g);
c =
   -0.0126
  203.3319
R2 =
    0.9985
```

依原始数据和拟合函数，可得到图9-7中图形。



467

图9-7 SiC的体积模量-温度函数的直线拟合

```
>> tfit = [ min(t) max(t)];
>> gfit = c(1)*tfit + c(2);
>> plot(t,g,'o',tfit,gfit,'-');
```

⊖ "Material Properties of a Sintered alpha-SiC," *J.Physical.Chem.Ref Data*, vol.26, pp.1195-1203, 1997.（另见www.nist.gov/srd/material.htm.）

较大的R^2值与图形一样，都表明直线对数据有较好的拟合。

9.1.5 显式非线性函数的多直线拟合

有些简单的非线性函数可以转化成线性函数。通过线性变换，可用线性最小二乘法对原始非线性数据进行曲线拟合。

线性变换的一个重要应用是指数式衰减（或增长）数据的拟合。假设例9.1中电路的各个测量值已知，则数据可描述为:

$$y = c_1 e^{c_2 x} \tag{9-19}$$

其中c_1为y的初始数据，$-1/c_2$为时间常数。方程（9-19）是关于参数c_1和c_2的非线性方程，因此c_1和c_2的值不能直接通过线性最小二乘过程求得。

方程（9-19）两边求对数，得

$$\ln y = \ln c_1 + c_2 x \tag{9-20}$$

另引入变量

$$v=\ln y \qquad \beta=\ln c_1 \quad \alpha=c_2 \tag{9-21}$$

则方程（9-19）变为

$$v=\alpha x+\beta$$

此方程为参数α和β的线性方程，对方程（9-19)进行线性最小二乘拟合的步骤为:

1. 将原始数据(x,y)用MATLAB变量存储。
2. 将数据(x,y)转变成(x,v)。
3. 将转变后的数据送入线性最小二乘拟合程序，此程序返回α和β。
4. 重新得到原始参数$c_1=e^{\beta}$, $c_2=\alpha$。

表9-1提供了对曲线拟合比较有用的一些线性转换的例子，其他的变换在文献[50、53]中给出。 468

表9-1 线性最小二乘拟合可用的数据转换

包含c_1和c_2的非线性函数	转化成$v=\alpha u+\beta$	
$y=c_1e^{c_2x}$	$v=\ln y$	$u=x$
	$\beta=\ln c_1$	$\alpha=c_2$
$y=c_1x^{c_2}$	$v=\ln y$	$u=\ln x$
	$\beta=\ln c_1$	$\alpha=c_2$
$y=c_1xe^{c_2x}$	$v=\ln(y/x)$	$u=x$
	$\beta=\ln c_1$	$\alpha=c_2$

读者要注意的是变换过程是对变换后数据（而非原始数据）的残差进行最小化，例如使用方程（9-21）中的变换来拟合指数式衰减数据，得到的c_1和c_2的值可使$\|\ln y-\ln y_{fit}\|_2$最小，而不是使$\|y-y_{fit}\|_2$最小。比起对方程（9-19）直接进行非线性最小二乘拟合，对变换数据拟合可以使小的$y_{fit}-y$值得到更小的权值。

例9.5 电容放电数据的曲线拟合

NMM工具箱中data目录下的`capacitor.dat`文件包含了例9.1中电容放电实验的电压-时间关系数据。

令$v=y$和$t=x$，下面的MATLAB语句将数据按方程（9-20）进行拟合。

```
>> load capacitor.dat;
>> t = capacitor(:,1);              % copy data for convenience
>> v = capacitor(:,2);
>> ct = linefit(t,log(v));          % Line fit to transformed data
>> c = [exp(ct(2));  ct(1)]         % Extract parameters from transformation
c =
     5.0000
   -10.0000
```

由于capacitor.dat中的数据是合成的，因此c_1和c_2为约整数。其中的电压-时间数据通过计算$v=5e^{-10t}$产生。当$|y/y_m|$的值与1相差很多时，对转换数据的拟合可以使数据点得到更大的权值。

例9.6　将噪声数据拟合成$y = 5xe^{-3x}$

本例中，答案在一开始就已知。为示范线性化坐标转换，要用一个已知函数$g(x)$来产生数据集(x,y)，然后向y值中加入人工噪声来仿真实验数据集中的测量误差。对噪声数据(x,y)进行最小二乘拟合，可以将此拟合与$g(x)$进行比较。由已知函数产生数据的过程，可以看出最小二乘拟合的质量。读者要记住，一般情况下，基础函数和噪声都是未知的。

469

程序清单9-2中的xexpfit函数使用两个向量x和y将数据拟合为形如$y=c_1x\exp(c_2x)$的方程。xexpfit函数只是转换数据，剩下的最小二乘计算留给linefit函数来做。虽然linefit函数的代码可复制到xexpfit函数中，但最好是重用linefit函数，这样可以利用包含在linefit函数中的改进和调试成果，也使xexpfit函数更加简洁和易读。改进和重用简洁代码模块是在扩充MATLAB工具箱过程中使用的典型策略，强烈建议读者使用。

程序清单9-2　xexpfit函数返回$y=c_1xe^{c_2x}$的最小二乘直线拟合的参数c_1和c_2

```
function c = xexpfit(x,y)
% xexpfit    Least squares fit of data to y = c(1)*x*exp(c(2)*x)
%
% Synopsis:  c = xexpfit(x,y)
%
% Input:     x,y = vectors of independent and dependent variable values
%
% Output:    c = vector of coefficients of  y = c(1)*x*exp(c(2)*x)

z = log(y./x);              % Natural log of element-by-element division
c = linefit(x,z);           % Fit is performed by linefit
c = [exp(c(2)); c(1); ];    % Extract parameters from transformation
```

程序清单9-3中的demoXexp函数向$g(x) = 5x\exp(-3x)$增加噪声（对数据进行干扰），以合成一个数据集。它有一个输入参数n，是将要产生的数据点的个数。demoXexp函数的m文件调用xexpfit函数来求解出c_1和c_2。噪声由内置函数rand产生。此函数的每次输出会随着自身的调用而改变[⊖]。故而，读者在自己的计算机上运行demoXexp的结果可能会与此处的结果稍有不同。

⊖ 在MATLAB 5及后续版本中，rand的伪随机输出依赖于机器状态；在MATLAB 4及早期版本中，伪随机数产生成器依赖于一个参数seed，每次MATLAB启动时就将seed参数重新设置为一个固定的值。

程序清单9-3　demoXexp函数对由$y = 5xe^{-3x}$生成并叠加噪声后的数据进行拟合，以示范xexpfit函数的使用

```
function demoXexp(n)
% demoXexp   Fit synthetic data to y = c(1)*x*exp(c(2)*x)
%
% Synopsis:  demoXexp(n)
%
% Input:     n = (optional) number of points to generate in the synthetic
%                data set;  Default: n = 200
%
% Output:    Fit coefficients are printed and a plot is created

if nargin<1,  n=200;  end

% --- Generate synthetic data and fit it
x0 = 0.01;                              %  Starting point ~= 0 avoids log(0)
noise = 0.05;                           %  Magnitude of noise
x = linspace(x0,2,n);
y = 5*x.*exp(-3*x);                     %  Create the true function y = g(x)
yn = y + noise*(rand(size(x))-0.5);     %  Noise multiplies random values v
                                        %  in the range -0.5 <= v <= 0.5
yn = abs(yn);                           %  Make sure all data are positive
c = xexpfit(x,yn);
fprintf('Fit parameters are c1 = %5.3f  c2 = %5.3f\n',c);

% --- Plot data and the fit function
xfit = linspace(min(x),max(x));
yfit = c(1)*xfit.*exp(c(2)*xfit);
if n<30,  s = 'v';  else s = '-';  end    %  Select symbol for original data
plot(x,y,s,x,yn,'o',xfit,yfit,'--');
xlabel('x');   ylabel('y');  legend('original','noisy','fit');
xmax = max(x);    ymax = max(y);
text(0.5*xmax,0.7*ymax,sprintf('c1 = %5.3f    c2 = %5.3f',c));
text(0.5*xmax,0.6*ymax,sprintf('%d points in synthetic data set',n));
```

图9-8描述了对一个长度$n = 200$的合成数据集使用xexpfit和demoxexp的结果。奇怪的是，c_1和c_2的拟合值并不分别接近各自的真实值5和−3。如果噪声不加到原始数据上（设demoXexp函数中的noise=0），拟合参数就会是精确的5和−3。c_1和c_2的真实值和拟合值出现差别是受两方面的影响。首先，这些噪声数据只是$f(x) = 5x\exp(-3x)$的所有随机扰动的一个有限样本。而且，由rand函数加入的噪声并非完全是随机的。语句yn=abs(yn)会破坏噪声的均匀度，因为此语句要求避免出现对负的y值取对数而产生的错误。

470
471

真实值和拟合值不一致的第二个原因是随着线性变换的使用而自然出现的。函数linefit会尽职地对已转换数据进行直线拟合。系数c_1和c_2描述了一条直线，它使$\ln(y/x)$而非y的残差的平方和最小。这样对由x0=0.1产生的数据集来说，拟合会使x较大的数据点的误差比x较小的数据点的误差获得更大的权值（参照习题2）。

从本例可以看出，最小二乘拟合的步骤并不复杂。拟合的结果很少有完美的，因此强烈推荐作出拟合图形进行检查。这并不是说曲线拟合的数据转换就可以绕过去。相反，在任何计算中，最好是对结果进行双重检查。由于MATLAB强大的画图能力，计算的图形检查变得非常简单，充分利用MATLAB的绘图能力是很明智的。

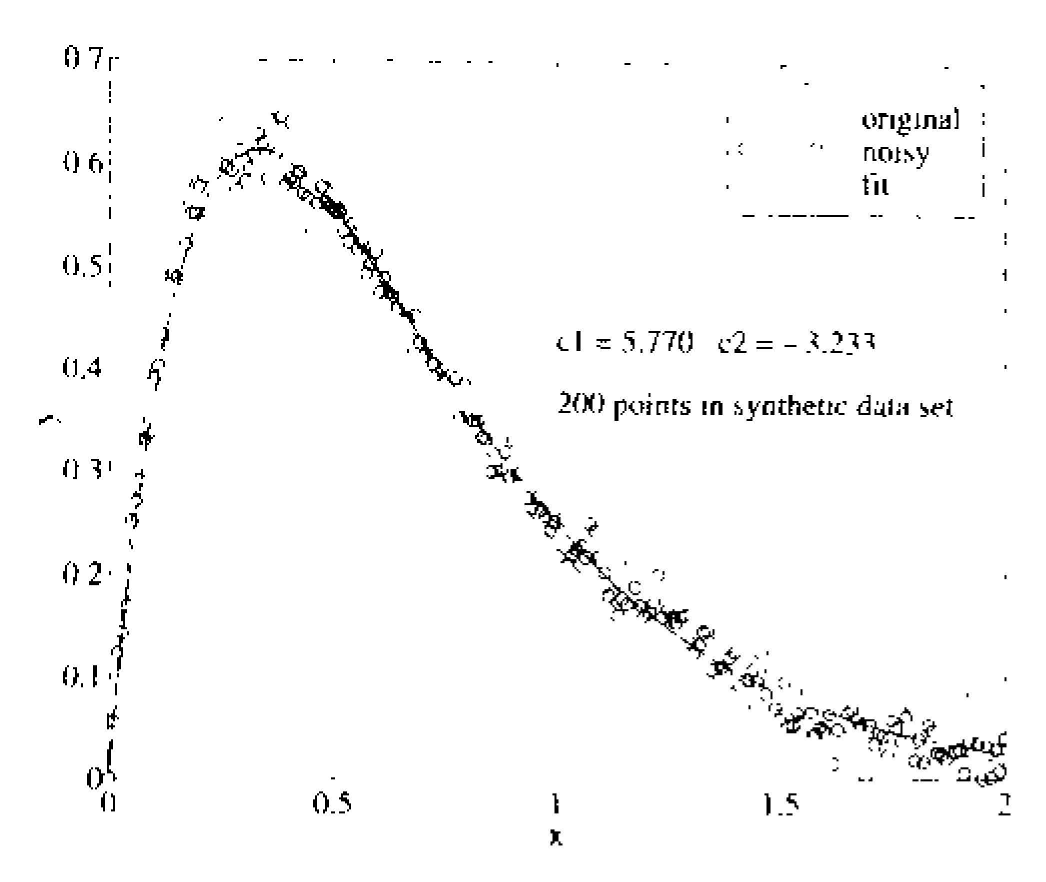


图9-8　对$y = 5xe^{-3x}$和噪声产生的数据集进行拟合

9.1.6　数据直线拟合小结

数据最小二乘直线拟合的本质特征：

1. 给出m个数据对：(x_i, y_i)，$i=1,\ldots,m$。
2. 拟合函数$y=F(x)=c_1x+c_2$有两个基本函数$f_1(x)=x$和$f_2(x)=1$。
3. 求出m个数据对的拟合函数可得到一个超定方程组：

$$\boldsymbol{Ac=y}$$

其中$\boldsymbol{c}=[c_1,c_2]^T$，$\boldsymbol{y}=[y_1,y_2,\ldots,y_m]^T$

$$\boldsymbol{A}=\begin{bmatrix} f_1(x_1) & f_2(x_1) \\ f_1(x_2) & f_2(x_2) \\ \vdots & \vdots \\ f_1(x_m) & f_2(x_m) \end{bmatrix}=\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_m & 1 \end{bmatrix}$$

[472]

4. 根据最小二乘法的定义，最好的拟合是使下列式子最小的c_1和c_2的值

$$\rho(c_1,c_2)=\|\boldsymbol{y}-F(x)\|_2^2=\|\boldsymbol{y}-\boldsymbol{Ac}\|_2^2$$

5. 将$\rho(c_1,c_2)$最小化可得到正规方程

$$(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{c}=\boldsymbol{A}^T\boldsymbol{y}$$

其中$(\boldsymbol{A}^T\boldsymbol{A})$是2×2矩阵，$\boldsymbol{A}^T\boldsymbol{y}$是包含两个元素的列向量。

6. 解出正规方程可得出最好的拟合直线的斜率c_1和截距c_2。

9.2　函数线性组合的最小二乘拟合

数据的最小二乘直线拟合是一般拟合过程的一个特殊例子。考虑下面的拟合函数：

$$y=F(x)=c_1f_1(x)+c_2f_2(x)+\cdots+c_nf_n(x) \tag{9-22}$$

其中每个基本函数$f_j(x)$, $j = 1,\ldots,n$, 为独立于系数c_j的任意函数。$f_j(x)$不需要是线性的。惟一的线性要求是$F(x)$必须是未确定参数c_j的线性组合，因此也就是$f_j(x)$的线性组合。拟合的目标是寻找$c_1,\ldots,c_n$，使给定数据和拟合函数之间的差最小。

更具体一点地说，首先假设将m个数据点拟合为三个基本函数：

$$y=F(x)=c_1f_1(x)+c_2f_2(x)+c_3f_3(x) \tag{9-23}$$

现在如果可找到一些c_j使$F(x)$通过所有的数据（也就是说，假设数据"排成一队"，使曲线拟合函数和插值式的功能一样）。那么，下面的m个方程都要严格满足：

473

$$\begin{gathered}c_1f_1(x_1)+c_2f_2(x_1)+c_3f_3(x_1)=y_1\\c_1f_1(x_2)+c_2f_2(x_2)+c_3f_3(x_2)=y_2\\\vdots\\c_1f_1(x_m)+c_2f_2(x_m)+c_3f_3(x_m)=y_m\end{gathered}$$

这等价于超定方程组

$$Ac=y \tag{9-24}$$

其中，

$$A=\begin{bmatrix}f_1(x_1)&f_2(x_1)&f_3(x_1)\\f_1(x_2)&f_2(x_2)&f_3(x_2)\\\vdots&\vdots&\vdots\\f_1(x_m)&f_2(x_m)&f_3(x_m)\end{bmatrix},\quad c=\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix},\quad y=\begin{bmatrix}y_1\\y_2\\\vdots\\y_m\end{bmatrix} \tag{9-25}$$

为推导方程（9-24），我们暂假设数据沿方程（9-23）给定的曲线分布（一般地，情形并非如此）。因为此处方程个数大于自由参数的个数($m>n$)，所以不可能得到方程（9-24）的精确解，最小二乘法提供了一种使 $\rho=\|y-Ac\|_2^2$ 最小的折衷解。如9.1.2节所述，ρ的最小值可由满足下列正规方程组中的向量c得到。

$$(A^TA)c=A^Ty \tag{9-26}$$

方程（9-25）中定义的A和c仅适于$n = 3$的情况。一般地，如方程（9-22）所示，有n个基本函数，故而矩阵A也有n个列，向量c中有n个元素

$$A=\begin{bmatrix}f_1(x_1)&f_2(x_1)&\cdots&f_n(x_1)\\f_1(x_2)&f_2(x_2)&\cdots&f_n(x_2)\\\vdots&\vdots&&\vdots\\f_1(x_m)&f_2(x_m)&\cdots&f_n(x_m)\end{bmatrix},\quad c=\begin{bmatrix}c_1\\c_2\\\vdots\\c_n\end{bmatrix},\quad y=\begin{bmatrix}y_1\\y_2\\\vdots\\y_m\end{bmatrix} \tag{9-27}$$

方程（9-26）的形式与基本函数的个数无关。

9.2.1 基本函数

方程（9-22）表示了拟合函数的一般形式。函数$F(x)$可以是函数关于c_j的任意线性组合，如

474

$$1,\ x,\ x^2,\ x^{2/3},\ \sin x,\ e^x,\ xe^{4x},\ \cos(\ln 25x)$$

都是有效的基本函数。另外，在下列函数中：

$$\sin(c_1x)\ ,\ e^{c_3x}\ ,\ x^{c_2}$$

只要c_i是拟合参数，它们就不是有效的基本函数。

三阶多项式拟合函数为

$$f(x)=c_1x^3+c_2x^2+c_3x+c_4 \tag{9-28}$$

其中基本函数为

$$x^3,\ x^2,\ x,\ 1$$

基本函数“1”使c_4成为了基本函数的常数项。方程（9-28)中的多项式是按x降幂排列进行书写的，这样就与MATLAB中工具箱函数polyfit和polyval的格式要求相一致，这两个函数将在9.2.4节中进行讨论。

9.2.2 通过求解正规方程组来进行最小二乘拟合

基本函数线性组合的最小二乘拟合的正规方程组与直线拟合的正规方程组形式相同。只有A的定义和未确定系数的个数改变了。为将数据拟合为n个基本函数的线性组合，需依照以下几个步骤：

1. 选择基本函数$f_j(x)$，$j=1,\ldots,n$。
2. 读入数据(x_i,y_i)，$i=1,\ldots,m$。
3. 计算出方程（9-27）中的A。
4. 解方程（9-26）得出系数向量c。

在下一个例子中，会用交互的MATLAB计算来示范数据的非线性函数最小二乘拟合的过程。下节将开发出自动实现这些计算的m文件函数。

例9.7 交互最小二乘拟合

最小二乘拟合问题只需几条MATLAB语句就可以很容易地解决。考虑数据(x,y)最小二乘拟合为

$$y=\frac{c_1}{x}+c_2x \tag{9-29}$$

的情况。

下表的数据存于NMM工具箱中data目录下的文件xinvpx.dat里

x	0.955	1.380	1.854	2.093	2.674	3.006	3.255	3.940	4.060
y	5.722	4.812	4.727	4.850	5.011	5.253	5.617	6.282	6.255

[475]

下列语句可以将表格中的数据进行曲线拟合，且拟合成为方程（9-29)的形式。

```
>> load xinvpx.dat;          % Read data from the "xinvpx.dat" file
>> x = xinvpx(:,1);          % Copy data into the x and y vectors
>> y = xinvpx(:,2);          %     (a convenience, only)
>> A = [1./x  x];            % Coefficient matrix of overdetermined system
>> c = (A'*A)\(A'*y)         % Solve normal equations
c =
    4.2596
    1.3008
```

注意到由`A = [1./x  x]`计算A是可行的，因为x是列向量。

下面，考虑不在原始数据集上，而是在x集合上的曲线拟合。例如，需要画一条平滑曲线通过原始数据。令x_f为待求解拟合函数所在x值的列向量，已知向量c中的系数，我们就可以使

用下列MATLAB向量语句来计算y。

```
>> xf = linspace(min(x),max(x));   % 100 points in range of original x data
>> yf = c(1)./xf + c(2)*xf;        % Evaluate the fit function at xf
```

因为x_f为列向量，我们可以将$y_f = F(x_f)$的计算写成

$$\begin{bmatrix} y_{f,1} \\ y_{f,2} \\ \vdots \\ y_{f,p} \end{bmatrix} = c_1 \begin{bmatrix} 1/x_{f,1} \\ 1/x_{f,2} \\ \vdots \\ 1/x_{f,p} \end{bmatrix} + c_2 \begin{bmatrix} x_{f,1} \\ x_{f,2} \\ \vdots \\ x_{f,p} \end{bmatrix}$$

对矩阵–向量乘积采用按列计算的算法（见7.2.2节算法7.1），我们可将上式写成

$$y_f = A_f\, c$$

其中，

$$A_f = \begin{bmatrix} 1/x_{f,1} & x_{f,1} \\ 1/x_{f,2} & x_{f,2} \\ \vdots & \vdots \\ 1/x_{f,p} & x_{f,p} \end{bmatrix}$$

于是，可求解出拟合函数$y_f = F(x_f)$，并用以下语句画图，结果如图9-9所示。

```
>> xf = linspace(min(x),max(x));   % 100 points in range of original x data
>> Af = [1./xf  xf];               % Eval basis fcns at xf as columns of A
>> yf = Af*c;                      % Evaluate the fit function at xf
>> plot(x,y,'o',xf,yf,'-');        % Plot original data and fit function
```

476

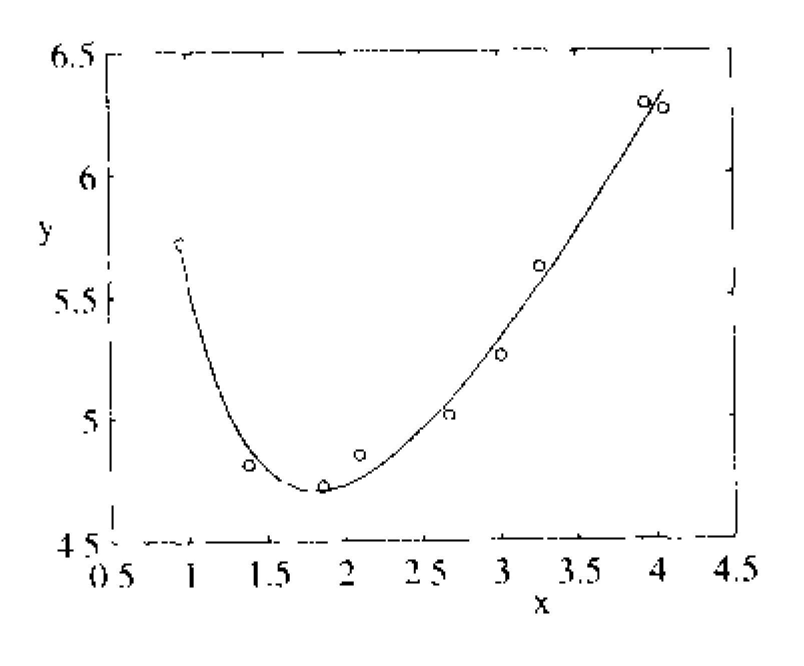


图9-9 例9.7中曲线拟合图

一般实现 最小二乘拟合过程可以总结成以下语句：

```
x = ...                 % Store x and y data in column vectors
y = ...
A = ...                 % Evaluate coefficient matrix of overdetermined system
c = (A'*A)\(A'*y)       % Solve normal equations
```

我们现在的目标是设计一个通用的m文件函数，它能在任意数据集及任意选出的基本函数上求得最小二乘拟合的系数。显然，数据(x,y)可以作为此函数的输入参数，但基本函数怎么确定呢？基本函数在求解矩阵A的过程中是非常必要的，求解A时也依赖于x中的数据。这里

选择的解法要求用户提供一个程序，该程序在给定任意列向量的情况下，计算出矩阵A。这样，用户不用提供矩阵A为输入，而是提供一个m文件的名字，此文件对任意x返回矩阵A。这个过程只用两行MATLAB代码来构造和求解正规方程组，因此有点曲折。但是它的优点是把要解决问题的说明与解决过程分离开来了，这样就可以使求解过程在某个地方进行更新——例如可添加更精巧复杂的曲线拟合分析。

477

程序清单9-4　函数fitnorm通过对任意函数组合求解正规方程组来得到数据的最小二线性乘拟合

```
function [c,R2,rout] = fitnorm(x,y,basefun)
% fitnorm   Least-squares fit via solution to the normal equations
%           Given ordered pairs of data, (x_i,y_i), i=1,...,m, fitnorm
%           returns the vector of coefficients, c_1,...,c_n, such that
%               F(x) = c_1*f_1(x) + c_2*f_2(x) + ... + c_n*f_n(x)
%           minimizes the L2 norm of y_i - F(x_i).
%
% Synopsis:  c         = fitnorm(x,y,basefun)
%            [c,R2]    = fitnorm(x,y,basefun)
%            [c,R2,r]  = fitnorm(x,y,basefun)
%
% Input:    x,y     = vectors of data to be fit
%           basefun = (string) name of user-supplied m-file that computes
%                     matrix A.  The columns of A are the values of the
%                     basis functions evaluated at the x data points.
%
% Output:   c = vector of coefficients obtained from the fit
%           R2 = (optional) adjusted coefficient of determination; 0 <= R2 <= 1
%                R2 close to 1 indicates a strong relationship between y and x
%           r  = (optional) residuals of the fit

if length(y)~= length(x);  error('x and y are not compatible');  end

A = feval(basefun,x(:));  %  Coefficient matrix of overdetermined system
c = (A'*A)\(A'*y(:));     %  Solve normal equations, y(:) is always a column
if nargout>1
  r = y - A*c;            %  Residuals at data points used to obtain the fit
  [m,n] = size(A);
  R2 = 1 - (m-1)/(m-n-1)*(norm(r)/norm(y-mean(y)))^2;
  if nargout>2,  rout = r;  end
end
```

程序清单9-4中的函数fitnorm采用了前面章节讨论的思想，对任意基本函数集合进行最小二乘曲线拟合。此函数可通过下面三种方法来调用：

```
c = fitnorm(x,y,basefun)
[c,R2] = fitnorm(x,y,basefun)
[c,R2,r] = fitnorm(x,y,basefun)
```

其中x和y是要进行拟合的数据组成的向量，*basefun*是用户自定义m文件的名字，它可以由基本函数计算出矩阵A。输出参数中，c是方程（9-22)中基本函数的系数组成的向量。R2是函数调整后的决定系数（下面要讨论），r是残差向量，$r = y - Ac$。

478

*basefun*所指定的m文件接收列向量x，并返回矩阵

$$A=\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix}$$

在fitnorm函数中，对*basefun*的调用通过内置的feval函数完成，即

```
A = feval(basefun,x( : ));
```

函数feval(...)中第二个参数位置上的子表达式x(:)是为了确保传给*basefun*的向量是一个列向量，即使传给fitnorm的向量是一个行向量。通常，计算基本函数的m文件只有一行MATLAB代码。例如，考虑下面计算例9.7的曲线拟合问题中矩阵A的函数：

```
function A = xinvpxBasis(x)
% xinvpxBasis  Matrix with columns evaluated with 1/x and x
A = [1./x  x];
```

这些语句要存入一个名叫xinvpxBasis.m的文件中。若用户想返回残差向量，则对fitnorm的调用为

```
>> [c,R2,r] = fitnorm(x,y,'xinvpxBasis');
```

若用户不需要R2的值，则对fitnorm的调用为：

```
>> c = fitnorm(x,y,'xinvpxBasis');
```

注意基本函数的个数n不需要在*basefun*程序中明确定义。另外，文件名的扩展名".m"不包括在*basefun*中的字符串值中。

对基本函数的简单集合，矩阵A的求解程序可以指定为一个嵌入函数对象。这样就不需要创建一个单独的m文件。在嵌入函数中定义xinvpxBasis， 调用fitnorm的语句为：

```
>> Afun = inline('[1./x x]');
>> [c,r] = fitnorm(x,y,Afun);
```

有关嵌入函数对象使用的其他方面将在例9.8中讨论。

在fitnorm中构造和求解正规方程组的语句为

```
c = (A'*A)\(A'*y( : ))
```

表达式y(:)保证A^Ty中的y为列向量。有关函数fitnorm的使用将在例9.8和例9.9中加以演示。

479

函数fitnorm的代码行中关键的只有四行，因此可以绕过函数调用，交互地进行拟合。或者，也可以将fitnorm的关键代码行合并到另一个求解矩阵A的程序中去。建议用户在具体应用时，选择适应相应问题的方法。若拟合在不同数据集和不同基本函数上展开时，使用fitnorm函数就很有利，请参见例9.9中所述的内容。

任意拟合函数的R^2统计量　9.1.4节中定义的R^2统计量也可以用于任意线性组合函数（方程 (9-17)和(9-18)中并没有明确地提到基本函数）的曲线拟合。计算任意基本函数组合的R^2统计量时，显然有

$$\hat{y}_i=\sum_{j=1}^{n}c_jf_j(x_i) \tag{9-30}$$

（也即是说，拟合函数已由已知（输入）数据求得）。

从技术上说，可以考虑调整系数后的决定函数，它的定义为

$$R^2_{\text{adjusted}} = 1 - \frac{m-1}{m-n-1}\frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2} \tag{9-31}$$

其中，m为数据点的个数，n为基本函数的个数（这种指定在本章通用）。$(m-1)/(m-n-1)$项说明了在求R^2时自由度的真实大小。对大多数曲线拟合问题来说，由于$m>>n$，故$(m-1)/(m-n-1)\approx 1$。考虑到m可能不远大于n，R^2_{adjusted}的值由fitnorm和fitqr计算得到。

一定要记住R^2或R^2_{adjusted}都只是拟合质量的反映。R^2或R^2_{adjusted}的值接近1说明因变量y与自变量x之间存在关系，R^2或R^2_{adjusted}的值较大不能保证拟合有意义。

例9.8　函数fitnorm的使用

函数fitnorm可以用来拟合例9.7中的数据。方程（9-29）中的基本函数使用NMM工具箱中fit目录下的xinvpxBasis.m文件求解。待拟合数据在NMM工具箱中data目录下的xinvpx.dat文件中。下面的交互命令用来载入数据并进行拟合。

480

```
>> load xinvpx.dat;         %  Contents of file stored in xinvpx matrix
>> x = xinvpx(:,1);         %  Copy 1st and 2nd columns into x and y
>> y = xinvpx(:,2);
>> [c,R2] = fitnorm(x,y,'xinvpxBasis')
c =
    4.2596
    1.3008


R2 =
    0.9817
```

为对拟合进行质量检验，画图时，要绘制出原始数据的点图和由原始数据所求解出的拟合函数的曲线。与例9.7类似，拟合函数由矩阵-向量乘积求得。

```
>> xf = linspace(min(x),max(x))'; %  100 points in range of original x data
>> Af = feval('xinvpxBasis',xf);  %  Eval basis fcns at xf as columns of A
>> yf = Af*c;                     %  Evaluate the fit function at xf
>> plot(x,y,'o',xf,yf,'-');
>> legend('data','fit',2);  xlabel('x');  ylabel('y');
```

结果图形就是前面的图9-9。下面的语句使用嵌入函数对象进行相同的计算：

```
>> ...
>> Afun = inline('[1./x(:)  x(:)]');
>> [c,R2] = fitnorm(x,y,Afun)
>> xf = linspace(min(x),max(x))';
>> Af = Afun(xf);
>> ...
```

注意，一旦Afun定义为一个嵌入函数对象，它就可以像标准的m文件函数一样使用。使用x(:)子表达式是考虑到向量x可能是一个行向量。在本例中，x(:)并不必要，因为x和y都是由xpinvx矩阵的列产生的，此矩阵内容存放在xpinvx.dat文件中。

使用矩阵来计算yf显露了fitnorm中求A过程的一个微弱优势。在任何曲线拟合问题中，基本函数都要（至少）使用两次：一次是求解出A来构造正规方程组，另一次是求解出拟合函数来画图。如果在计算A时，基本函数通过定义外部程序或嵌入函数对象来构造，那它们就只需要使用一次。使用这种思想自动计算曲线拟合过程的问题将在习题15中进一步探讨。

例9.9 导热性数据的曲线拟合

例9.2中的实验数据描述了铜在接近绝对零度下的热导率k的变化。我们现在开始求这些数据的曲线拟合。三种不同铜样品的k和T的测量值分别存放在NMM工具箱data目录下的文件cucon1.dat、cucon2.dat和cucon3.dat中。数据的图示参照图9-3。

根据k随着T改变的物理机理，可提出$k(T)$的数学模型如下

$$k(T)=\frac{1}{\frac{c_1}{T}+c_2T^2} \tag{9-32}$$

此方程关于c_1和c_2并不是线性的。为了用线性最小二乘法将数据拟合成这个函数，定义

$$\gamma(T)=\frac{1}{k(T)}=\frac{c_1}{T}+c_2T^2 \tag{9-33}$$

并用$\gamma-T$关系来代替$k-T$关系。方程（9-33）中的用来定义矩阵A的基本函数存放在cuconBasis1.m文件中，此文件存放在NMM工具箱里。另外，基本函数也可以使用嵌入函数对象来求得，这将在下面进行演示。

对此数据进行交互输入拟合的关键步骤如下：

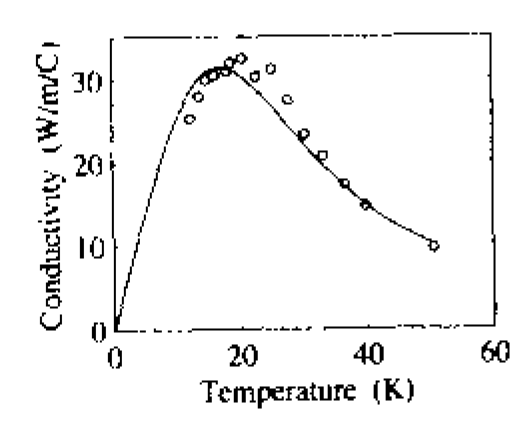
```
>> fun1 = inline('[1./t  t.^2]');                % x must be a column vector
>> [t,k] = loadColData('cucon1.dat',2,0,2);    % read data into t and k
>> [c,r] = fitnorm(t,1./k,fun1);               % perform the fit
>> fprintf('    %e\n',c)                        % print coefficients
   3.616821e-01
   3.661922e-05
```

首先，将矩阵A（由基本函数实现）的计算程序定义为一个嵌入函数对象。然后，用loadColData函数将数据从指定的文件中读出，此函数是NMM工具箱中的一个实用程序，用来从一个包含列标题的文本中读取数据（参见utils目录）。函数fitnorm用来解正规方程组，求解出系数。

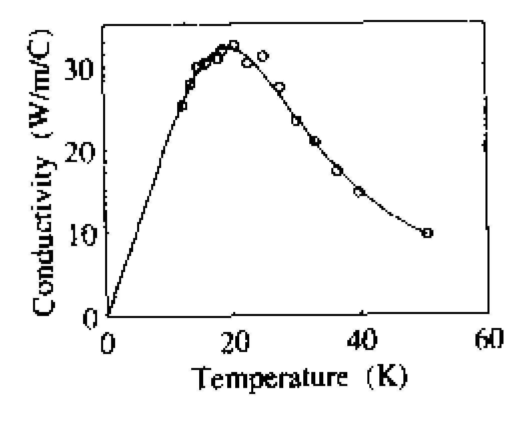
下面的语句求解出t范围内100个点上的拟合，并画图比较拟合结果与原始数据：

```
>> tf = linspace(min(t),max(t))';
>> Af = feval(fun1,tf);                % evaluate basis fcns for gamma(t)
>> kf = 1./(Af*c);                     % k = 1/gamma
>> plot(t,k,'o',tf,kf,'-')
```

图9-10a描述了原始数据和对方程（9-32)的拟合。



a) cucon1.dat中的数据拟合成方程（9-33）



b) cucon1.dat中的数据拟合成方程（9-34）

图9-10 cucon1.dat文件中的导热性数据的最小二乘拟合

向函数$k(T)$的分母增加一个T的线性项，可以在相同数据上得到更好的曲线拟合，即

482

$$\gamma_2(T)=\frac{1}{k(T)}=\frac{c_1}{T}+c_2T+c_3T^2 \tag{9-34}$$

γ_2的基本函数由一个嵌入函数对象来实现：

```
>> fun2 = inline('[1./t  t  t.^2]');
```

图9-10b表示方程（9-34)的曲线拟合。此图的质量改进可通过残差值的减小反映出来：方程9-33的拟合中 $\|y-Ac\|_2=9.6\times10^{-3}$，而方程（9-34)的拟合中 $\|y-Ac\|_2=4.5\times10^{-3}$。

程序清单9-5中的conductFit函数对导热性数据自动进行最小二乘拟合。此函数用来比较两个不同基本函数集合的拟合适合程度，这些基本函数用来拟合多个$k-T$数据集。除了进行拟合，函数conductFit还为拟合函数和原始数据画图。此函数只有一个输入，即包含$k-T$数据的文件名。使用conductFit与文件cucon2.dat和cucon3.dat一起可以画出图9-11。

函数conductFit也打印拟合系数（c_j）和 $\|y-Ac\|_2$ 的值。为节省空间，这里不打印曲线拟合系数的值。下面的表格摘录了三个数据集上两个不同基本函数的 $\|y-Ac\|_2$ 值：

数据集	方程（9-33）		方程（9-34）	
	$\|y-Ac\|_2$	$R^2_{adjusted}$	$\|y-Ac\|_2$	$R^2_{adjusted}$
cucon1.dat	0.01150	0.973	0.00387	0.997
cucon2.dat	0.00956	0.979	0.00450	0.994
cucon3.dat	0.00872	0.966	0.00673	0.977

483

程序清单9-5 函数conductFit用来对铜在低温下的导热性数据进行拟合

```
function conductFit(fname)
% conductFit  LS fit of conductivity data for Copper at low temperatures
%
% Synopsis:  conductFit(fname)
%
% Input:  fname   = (optional, string) name of data file;
%                   Default: fname = 'conduct1.dat'
%
% Output:  Print out of curve fit coefficients and a plot comparing data
%          with the curve fit for two sets of basis functions.

if nargin<1,  fname = 'cucon1.dat';   end   %  Default data file

% --- define basis functions as inline function objects
fun1 = inline('[1./t  t.^2]');              %  t must be a column vector
fun2 = inline('[1./t  t  t.^2]');

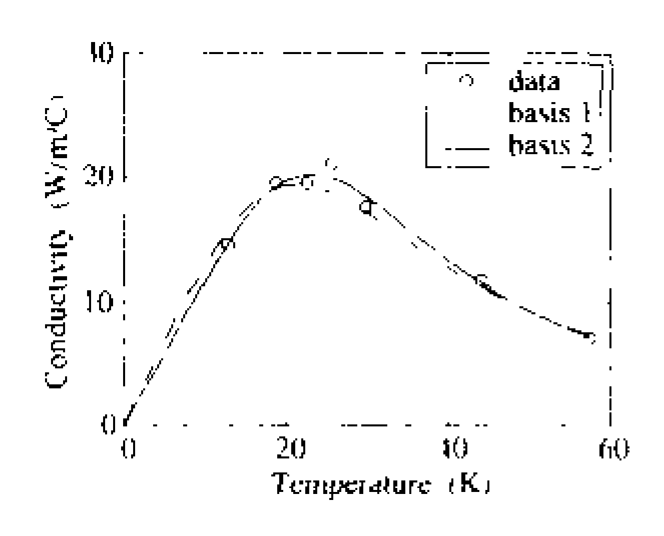
% --- read data and perform the fit
[t,k] = loadColData(fname,2,0,2);         %  Read data into t and k
[c1,R21,r1] = fitnorm(t,1./k,fun1);       %  Fit to first set of bases
[c2,R22,r2] = fitnorm(t,1./k,fun2);       %  and second set of bases

% --- print results
fprintf('\nCurve fit to data in %s\n\n',fname);
fprintf(' Coefficients of      Basis Fcns 1        Basis Fcns 2\n');
```

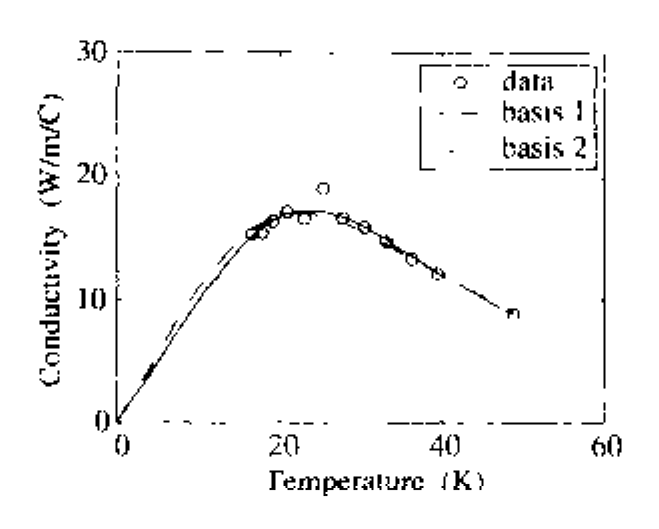
```
fprintf('      T^(-1)      %16.9e   %16.9e\n',c1(1),c2(1));
fprintf('      T           %16.9e   %16.9e\n',0,c2(2));
fprintf('      T^2         %16.9e   %16.9e\n',c1(2),c2(3));
fprintf('\n    ||r||_2        %12.5f        %12.5f\n',norm(r1),norm(r2));
fprintf('      R2          %12.5f        %12.5f\n',R21,R22);

% --- evaluate and plot the fits
tf = linspace(0.1,max(t))';       %  100 T values: 0 < t <= max(t)
Af1 = feval(fun1,tf);             %  A matrix evaluated at tf values
kf1 = 1./ (Af1*c1);               %  Af*c is column vector of 1/kf values
Af2 = feval(fun2,tf);
kf2 = 1./ (Af2*c2);
plot(t,k,'o',tf,kf1,'--',tf,kf2,'-');
xlabel('Temperature  (K)');   ylabel('Conductivity  (W/m/C)');
legend('data','basis 1','basis 2');
```

484



a) 文件cucon2.dat中的数据对方程（9-33）和方程（9-34）的拟合



b) 文件cucon3.dat中的数据对方程（9-33）和方程（9-34）的拟合

图9-11　文件cucon2.dat和cucon3.dat中的导热性数据的最小二乘拟合

个数据集中，由方程（9-34）产生的$\|y-Ac\|_2$值要比方程（9-33）产生的$\|y-Ac\|_2$值小，方程（9-34）得到的R^2的值也相应地比方程（9-33）得到的R^2的值要大，于是我们推断方程（9-34）对数据的拟合比方程（9-33）要好。拟合函数相差最大的是cucon1.dat集合，这从图9-10中也可以明显地看出来。

9.2.3　用QR分解法进行最小二乘逼近（拟合）

正规方程法常用来解决最小二乘问题，但是这不是求解c使$\|y-Ac\|_2$值最小的惟一方法。最小二乘问题也可由QR分解和奇异值分解（*Singular Value Decomposition*，SVD）来解决。这部分中，我们将描述QR分解。通过一些实现最小二乘法的MATLAB内置命令（如polyfit），使用QR分解法来求解超定方程组。

在严格的算术意义上，通过正规方程法、QR分解法或SVD法来解最小二乘问题没什么区别[⊖]。这些方法的主要不同点在于计算所花费的浮点操作次数和数值上的稳定性。

回顾8.3.3节，数值算法的稳定性与对输入数据扰动的受影响程度有关。对最小二乘问题来说，扰动是由测量数据(x, y)的不精确引起的，而这些数据要用来形成矩阵A和超定方程组

485

⊖ 如果A是一个$m\times n$矩阵$(m>n)$，且$\text{rank}(A)=n$，那么线性最小二乘问题就有使$\|b-Ax\|_2$最小的惟一解（例见[28，6.3节]或[32，5.3节]），若无舍入，所有解最小二乘问题的数值算法都得到此惟一解。

的右边向量。测量数据很小的改变（如重复相同的物理实验的测量数据）不应该对拟合系数的影响很大。因此，对最小二乘拟合来说稳定性是非常重要的。用QR分解法和SVD法求解最小二乘问题时比用正规方程组更稳定。

若以CPU时间和内存来度量，求解最小二乘问题的最有效方法是正规方程法。但是，对很多实际问题求解时，CPU时间和内存上的节省并不重要。类似地，若计算使用双精度，基于QR分解法和SVD法的最小二乘解法在稳定性上的提高也不会有很大意义。既然采用正规方程所需计算时间和内存的节省很显著，而QR分解法或SVD法在稳定性上也有明显提高，那么，哪个方法最值得推荐呢？

使用QR分解法来解决最小二乘问题是很有道理的。由于正规方程组是病态的，可能会导致损失有效位数而没有任何的警告，也即是说，问题的求解过程会直奔终点，然后用户会毫无疑问地接受这个结果。所以，最好选用QR分解法，因为使用它可以降低有效位数的损失，虽然并不能完全排除。另外，如果使用QR分解法会导致运行时间长或大量占据内存，用户会清楚地看到这些现象。那样的话，就可以选用更高效的正规方程法。幸运的是，对MATLAB用户来说在这些方法之间切换非常简单，将NMM工具箱中的`fitnorm`和`fitqr`两个m文件作一下比较就非常明显了。

在下面的部分中，我们首先讨论QR分解的属性以及它与最小二乘问题的关系，然后我们讨论使用QR分解来求解最小二乘问题的MATLAB技术。这里并不描述如何得到QR分解的细节。最后讲述了使用QR分解来求解最小二乘问题的MATLAB技术，并对最小二乘求解中反斜杠操作符的使用进行了演示。

通过考虑对不共线的三个点的直线拟合，可以看出QR分解法的精髓。我们选择三个具体点（1，1），（2，3）和（3，4）。这个问题的超定方程组是：

$$\begin{bmatrix}1 & 1\\2 & 1\\3 & 1\end{bmatrix}\begin{bmatrix}c_1\\c_2\end{bmatrix}=\begin{bmatrix}1\\3\\4\end{bmatrix}$$

高斯消去法初探　为了导出为何要使用QR分解，我们将首先对上面的方程组进行高斯消去。虽然高斯消去法可以用在带有矩形系数矩阵的方程组上，但我们将会看到，所得结果并非最小二乘解。

486

建立增广方程组（比较8.2.3节），并用不选主元的高斯消去法，可得：

$$\left[\begin{array}{cc|c}1 & 1 & 1\\2 & 1 & 3\\3 & 1 & 4\end{array}\right]\rightarrow\left[\begin{array}{cc|c}1 & 1 & 1\\0 & -1 & 1\\0 & -2 & 1\end{array}\right]\rightarrow\left[\begin{array}{cc|c}1 & 1 & 1\\0 & -1 & 1\\0 & 0 & -1\end{array}\right]$$

消去过程的最后一步，矩阵的最末一行表示$0c_1+0c_2=-1$，方程无解，也即是说，c_1和c_2的值可以满足前两个方程，但是不能满足第三个方程。这也和经过不共线三点不可能画一条直线的观察结果相一致。

回顾8.4.1节，高斯消去法可以通过自左乘一个消去矩阵得到。对现在的A，我们有

$$MA=\begin{bmatrix}1 & 0 & 0\\-2 & 1 & 0\\1 & -2 & 1\end{bmatrix}\begin{bmatrix}1 & 1\\2 & 1\\3 & 1\end{bmatrix}=\begin{bmatrix}1 & 1\\0 & -1\\0 & 0\end{bmatrix}$$

高斯消去法不能产生最小二乘解的原因是乘以M不符合残差的L_2范数

$$\|M(y-Ac)\|_2 \neq \|y-Ac\|_2$$

这在求最小二乘解的过程中很重要，因为目标是找到向量c使残差最小。与高斯消去法不同的是，QR分解可以在使用时遵循L_2范数，这也就是要使用QR分解法，而不是高斯消去法来解决最小二乘问题的原因。注意对$n \times n$矩阵A和$n \times 1$向量x和b，用（适当的）消去矩阵M左乘很有用，因为解方阵方程组$MAx = Mb$与解方阵方程组$Ax = b$是一样的。

QR分解 回顾8.4.1节，高斯消去法相当于将矩阵A分解成两个三角矩阵L和U的乘积。A的另一种分解方法就是QR分解。

令A为$m \times n$矩阵（$m>n$）。A的QR分解是

$$A = QR \tag{9-35}$$

其中Q是$m \times m$的正交矩阵（*orthogonal matrix*），R是$m \times n$的上三角矩阵。如果A是4×2矩阵，它的QR分解就形如

487

$$\begin{array}{ccc} A & = \quad Q & R \\ \begin{bmatrix} \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \\ \bullet & \bullet \end{bmatrix} & = \begin{bmatrix} \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet \end{bmatrix} & \begin{bmatrix} \bullet & \bullet \\ 0 & \bullet \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \\ 4\times 2 & 4\times 4 & 4\times 2 \end{array}$$

其中“•”表示非零值。既然Q为正交矩阵，就有$Q^TQ = I$，且$Q^{-1} = Q^T$。矩阵R为上三角矩阵。一般地，既然R为$m \times n$，那么它最后$m-n$行就全部是零。

将矩阵-矩阵按列乘，可以深入理解QR分解。（参考7.2.2节和图7.8）令$q_{(j)}$、$r_{(j)}$、$a_{(j)}$分别为Q、R、A的第j列，并写出Q和R的乘积如下：

$$\left[\begin{array}{c|c|c|c} q_{(1)} & q_{(2)} & \cdots & q_{(m)} \end{array}\right]\left[\begin{array}{c} \cdots r_{(j)} \cdots \end{array}\right] = \left[\begin{array}{c} \cdots a_{(j)} \cdots \end{array}\right]$$

以此来看矩阵-矩阵乘积，例如A的第三列为

$$r_{1,3}q_{(1)} + r_{2,3}q_{(2)} + r_{3,3}q_{(3)} = a_{(3)}$$

换言之，$a_{(3)}$就是Q的前三列的线性组合。一般地，因为$r_{i,j} = 0(i>j)$，那么就只有Q的1到j列需要参与计算A的j列。因此，整个矩阵A（$m \times n$，$m>n$）只需用Q的前n列就可以表示出来。这里假设A的秩为n（也就是说，A的所有n个列都线性无关）。若$\text{rank}(A) = k < n$，那么只有Q的前k列需要。R的最后$m-n$行必须为零，因为Q的列正交，所以也线性无关。

因为Q为$m \times m$矩阵，各列正交，那么Q的前n列就形成A的列空间的基（即$\text{range}(A)$）。Q的最后$m-n$列是$\text{range}(A)$以外的R^m子空间的基，这个子空间就叫$\text{range}(A)$的正交补（*orthogonal complement*），记为$\text{range}(A)^{\perp}$。下面的讨论中，Q的最后$m-n$列不参与用QR分解法进行最小二乘逼近的计算。

设$\hat{Q}$为Q前n列组成的矩阵，Q_c为剩余列组成的矩阵（“c”表示这些列形成了$\text{range}(A)^{\perp}$的基），我们将Q写成分块矩阵的形式：

488

$$Q=\begin{bmatrix}\hat{Q} & Q_{\perp}\end{bmatrix} \tag{9-36}$$

相应的R的分块矩阵形式为:

$$R=\begin{bmatrix}\hat{R}\\0\end{bmatrix} \tag{9-37}$$

其中 $\hat{R}$ 代表矩阵R的非零上三角$n\times n$方阵，0代表$(m-n)\times n$的零矩阵。

因为只有Q的前n列需要用R中的系数来产生A，我们就可以在计算QR分解过程中省很多力气。所谓的A的简略QR分解（*economy-size QR factorization*）就是

$$A=\hat{Q}\hat{R} \tag{9-38}$$

完全QR分解和简略分解之间只有一个实质上的差别，就是完全分解包含了Q的额外的$m-n$列，这些列形成了range$(A)^{\perp}$的基。

我们不讨论计算矩阵QR分解的各种算法。相反，我们依靠MATLAB中内置的qr函数对给定矩阵A计算Q和R。想深入了解的读者可详细参考Datta[12]和Watkins[78]，这些是中级讲解水平的参考书。Gill等[28]、Golub and Van Loan[32] 中有更简洁的描述，但这需要读者更深厚的知识。Stewart[69]在较高层次上提供了完整的讲解。Trefethen and Bau[76]从一种直观的数学角度讲解了QR分解和最小二乘问题。Demmel[15]用现今的数值线性代数规范讨论了最小二乘的QR分解实现。

内置的qr函数返回一个矩阵的QR分解。调用它有两种方法:

```
[Q,R] = qr(A)
[Q̂,R̂] = qr(A,0)
```

其中，Q和$\hat{Q}$ 是正交矩阵，R和 $\hat{R}$ 是上三角矩阵。$[Q,R]$ = qr(A)形式返回完全的QR分解（即，若A是$m\times n$，那么Q是$m\times m$，R是$m\times n$.）。$[\hat{Q},\hat{R}]$ = qr$(A,0)$形式返回简略QR分解，其中 $\hat{Q}$ 和 $\hat{R}$ 是方程（9-38)中的 $\hat{Q}$ 和 $\hat{R}$ 矩阵。

例9.10　函数qr的使用

本例将使用内置的qr函数来求解一个3 × 2矩阵的QR分解，此矩阵来自前面讨论的三数据点最小二乘拟合问题的超定方程组。即

489

$$A=\begin{bmatrix}1 & 1\\2 & 1\\3 & 1\end{bmatrix}$$

对此矩阵使用内置的qr函数得

```
>> A = [1 1; 2 1; 3 1];
>> [Q,R] = qr(A)
Q =
   -0.2673    0.8729    0.4082
   -0.5345    0.2182   -0.8165
   -0.8018   -0.4364    0.4082
R =
   -3.7417   -1.6036
         0    0.6547
         0         0
```

虽然A中的元素为整数，但Q和R中的元素是浮点数，这里只显示前四位有效位。在舍入

范围内，分解是成功的：

```
>> A-Q*R
ans =
   1.0e-15 *
    0.5551   -0.2220
    0.2220         0
    0.4441         0
```

同样Q也是正交的：

```
>> Q'*Q - eye(3) =
ans =
   1.0e-15 *
         0   -0.0541   -0.0663
   -0.0541    0.2220    0.2034
   -0.0663    0.2034         0
```

*A*的简略QR分解（参照方程（9.38））为：

```
>> [Q0,R0] = qr(A,0)
Q0 =
   -0.2673    0.8729
   -0.5345    0.2182
   -0.8018   -0.4364

R0 =
   -3.7417   -1.6036
         0    0.6547
```

490

正如期望的那样，Q0和Q的前两列相同，R0和R的前两行相同。在*A*的简略分解中，Q0和R0包含了重新构造*A*的所有信息。

```
>> norm(A-Q0*R0,inf)
ans =
   7.7716e-16
```

使用QR分解来求最小二乘解 一个向量乘以一个正交矩阵，它的L_2范数不变，即

$$\|Qx\|_2 = \left[(Qx)^T(Qx)\right]^{1/2} = \left[(x^TQ^T)(Qx)\right]^{1/2} = \left[x^T(Q^TQ)x\right]^{1/2}$$
$$= \left[x^Tx\right]^{1/2} = \|x\|_2$$

这个特征使QR分解适合于最小二乘问题求解。由Q的特性，使$\|y-Ac\|_2$最小的最小二乘曲线拟合问题可以转换成

$$\|y-Ac\|_2 = \|Q^T(y-Ac)\|_2 = \|Q^Ty-Q^TAc\|_2 = \|Q^Ty-Rc\|_2$$

其中右边最后一个表达式是将*A*代替为*QR*推导出的。以上可总结为：

> **求超定方程组$Ac=y$的最小二乘解相当于求$\|Q^Ty-Rc\|_2$的最小值，其中$A=QR$是A的QR分解**

之所以将最小二乘问题写成这种形式，是因为一旦*A*分解成*Q*和*R*，那么使$\|Q^Ty-Ac\|_2$最小的*c*就很容易计算。

让我们回到三个不共线数据点的最小二乘直线拟合问题。为了使概念清楚，我们要用符号来代替数值（*Q*和*R*的元素不是整数，使用浮点数值不易看到式子的变换）。写出三个点最

小二乘拟合的$\|Q'y - Rc\|_2$，得

$$\|Q'y - Rc\|_2 = \left\| \begin{bmatrix} q_{11} & q_{21} & q_{31} \\ q_{12} & q_{22} & q_{32} \\ q_{13} & q_{23} & q_{33} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} - \begin{bmatrix} r_{11} & r_{12} \\ 0 & r_{22} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} \right\|_2$$

$$= \left\| \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} - \begin{bmatrix} r_{11}c_1 + r_{12}c_2 \\ r_{22}c_2 \\ 0 \end{bmatrix} \right\|_2 \tag{9-39}$$

491

其中$z = Q'y$。最小二乘解就是使方程（9-39）右边标量最小的向量c。现在我们作一个重要的观察：选择c_1和c_2不能改变方程（9-39）中向量$z - Rc$第三个部分的大小，但是，我们可以选择c_1和c_2使前两个元素为零。换言之，方程

$$\begin{bmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \tag{9-40}$$

的解使可以$\|r\|_2 = \|Q'y - Rc\|_2$的值最小。另外，从方程（9-39）可知，精确地解方程（9-40），就有$\|r\|_2 = \|z_3\|$。

将前面的推导转换成MATLAB语言，可以得到下列用QR分解法求最小二乘问题的步骤：

```
>> x = ...;  y = ...;   A = ...;   % set up the overdetermined system
>> [m,n] = size(A);
>> [Q,R] = qr(A);                  % full QR factorization
>> z = Q'*y;
>> c = R(1:n,1:n)\z(1:n)
```

最后一步求R的前n行所构成简化方程组的解，此方程组仅包含了方程（9-36）到方程（9-38）中定义的简略QR分解。将$Q'y - Rc$写成分块矩阵的形式，这样可更清楚明了地体现$Q'y - Rc$与简略QR分解的联系

$$\|Q'y - Rc\|_2 = \left\| \begin{matrix} \tilde{Q}'y - \tilde{R}c \\ Q_r'y - 0c \end{matrix} \right\|_2 = \left\| \begin{matrix} \tilde{z} - \tilde{R}c \\ z_r - 0c \end{matrix} \right\|_2 = \|\tilde{z} - \tilde{R}c\|_2 + \|z_r\|_2$$

其中，最后的等号后面是直接计算$\|\cdot\|_2$的值。因此，残差的大小包括两部分：$\|\tilde{z} - \tilde{R}c\|_2$，它可以设定为零，通过解$\tilde{R}c = \tilde{z}$求出$c$；以及$\|z_r\|_2 = \|Q_r^T y\|_2$，后者不会因为任何$c$值而得到简化。

简略QR分解给出了下列算法来求解最小二乘问题：

算法9.1　用QR分解法求最小二乘解

[m,n]=size(A)

Factor: $A = \tilde{Q}\tilde{R}$　　　　（简略分解）

$\tilde{z} = \tilde{Q}'y$

solve $\tilde{R}c = \tilde{z}$

$\tilde{R}$是上三角矩阵，因此最后一步要通过向后代入才能得到。算法9.1适用于下例介绍的小规模的拟合问题。

492

例9.11 用QR分解法进行最小二乘直线拟合

用QR分解法将三个数据对(1,1)，(2,3)和(3,4)拟合成直线$y = c_1x + c_2$。简略QR分解可由内置qr函数得到：

```
>> A = [1 1; 2 1; 3 1];    % matrix of overdetermined system
>> y = [1; 2; 3];
>> [Q0,R0] = qr(A,0);      % Economy-size QR factorization of A
>> z0 = Q0'*y
z0 =
   -5.0780
   -0.2182

>> c = R0\z0
c =
    1.5000
   -0.3333
```

有兴趣的读者可以通过linefit函数或手工计算来证明，$c = [3/2, 1/3]^T$就是三点最小二乘拟合问题的解。

用QR分解法和MATLAB反斜杠进行最小二乘计算 如8.4.3节所述，当MATLAB要解析形如c = A\y的表达式时，首先要检查A的形状。若A为$m \times n$矩阵（$m>n$），MATLAB就自动地对A应用QR分解法，求解出使$\|y - Ac\|_2$最小的c。因此，$Ac=y$的最小二乘解可由以下MATLAB语句得到：

```
>> A = ...          % Define A and y for the overdetermined system
>> y = ...
>> c = A\y          % Solve the least-squares problem
```

显然，使用QR分解法非常简便，因为MATLAB很善于处理线性代数问题。遇到类似A\y的表达式且A不是方阵时，MATLAB就假设用户希望得到最小二乘"解"。

使用反斜杠操作符节省的不仅仅是键盘输入。由QR分解法得到最小二乘解时，矩阵Q不必明确地计算出来，它只在$\hat{z} = \hat{Q}'y$的计算中出现。矩阵的QR分解涉及一系列预备性（初始）的正交矩阵的计算，由于这些矩阵已知，故可以直接将它们应用到y中（见[77, 7.2.3节]中关于这方面计算的MATLAB程序），这节省了存储Q的内存，同时也明显降低了求解过程中必需的浮点操作次数。

493

用QR分解法实现曲线拟合 现在，对创建一个用QR分解法进行最小二乘拟合的通用的m文件来说，所需要的部分都已经具备了。这个m文件函数就是程序清单9-6中的fitqr函数。函数fitqr与fitnorm几乎一样，不同的地方只有一行（即向量c的求解上）：

```
fitnorm:  c = (A'*A)\(A'*y(:));
fitqr:    c = A\y(:);
```

程序清单9-6 fitqr函数使用QR分解法对任意函数组合进行数据的线性最小二乘拟合

```
function [c,R2,rout] = fitqr(x,y,basefun)
% fitqr  Least-squares fit via solution of overdetermined system with QR
%        Given ordered pairs of data, (x_i,y_i), i=1,...,m, fitqr
%        returns the vector of coefficients, c_1,...,c_n, such that
%            F(x) = c_1*f_1(x) + c_2*f_2(x) + ... + c_n*f_n(x)
%        minimizes the L2 norm of y_i - F(x_i).
```

```
%
% Synopsis:  c          = fitqr(x,y,basefun)
%            [c,R2]     = fitqr(x,y,basefun)
%            [c,R2,r]   = fitqr(x,y,basefun)
%
% Input:     x,y     = vectors of data to be fit
%            basefun = (string) name of user-supplied m-file that computes
%                      matrix A.  The columns of A are the values of the
%                      basis functions evaluated at the x data points.
%
% Output.    c = vector of coefficients obtained from the fit
%            R2 = (optional) adjusted coefficient of determination; 0 <= R2 <= 1
%                 R2 close to 1 indicates a strong relationship between y and x
%            r  = (optional) residuals of the fit

if length(y)~= length(x);  error('x and y are not compatible');  end

A = feval(basefun,x(:)); %  Coefficient matrix of overdetermined system
c = A\y(:);              %  Solve overdetermined system with QR factorization
if nargout>1
  r = y - A*c;           %  Residuals at data points used to obtain the fit
  [m,n] = size(A);
  R2 = 1 - (m-1)/(m-n-1)*(norm(r)/norm(y-mean(y)))^2;
  if nargout>2,  rout = r;  end
end
```

494

函数fitqr与函数fitnorm使用相同的基本函数程序，可以很容易地用fitqr代替fitnorm来改变最小二乘曲线拟合问题求解的数值算法。对于良性问题，这两种方法得到的结果几乎一样。在下一节的开始就介绍了为什么推荐使用fitqr函数。

9.2.4 多项式曲线拟合

可以用函数fitqr或fitnorm将数据拟合为多项式，这需要构造m文件来计算出一个k阶多项式的单项式基本函数1，x，x^2...x^k。但这里并不采纳这种方法，单项式基本函数的规则性可以简化算法，从而能更简洁方便地为拟合构造超定方程组。

考虑数据拟合为二次方程式($y = c_1x^2 + c_2x + c_3$)的情况。给定已知数据的列向量x和y，拟合的超定方程组为(另见方程（9-24))

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m^2 & x_m & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \tag{9-41}$$

利用MATLAB的数组操作符，二阶多项式的拟合可以用以下几行代码来建立并解决：

```
>> x = ...       %  x and y are column vectors of known data
>> y = ...
>> A = [ x.^2  x  ones(size(x)) ];  %  Overdetermined coefficeint matrix
>> c = A\y;                         %  Least squares solution
```

polyfit函数 内置polyfit函数可使用刚才描述的过程来进行n阶多项式的最小二乘曲线拟合

$$F(x)=p_1x^n+p_2x^{n-1}+\cdots+p_nx+p_{n+1}$$

函数polyfit构造出系数矩阵，并由QR分解得出超定方程组的最小二乘解，它可以用下面两种方法调用：

495

```
p = polyfit(x,y,n)
[p,S] = polyfit(x,y,n)
```

x和y是定义待拟合数据的向量，n是多项式的阶数。多项式依x降幂排列，其系数返回后放在向量p中。第二个可选的输出参数S是一个数据结构，它包含三个元素：S.R是简略QR分解中的矩阵R；S.df是拟合系数的自由度数；S.normr是$\|\mathbf{y}-\boldsymbol{A}\mathbf{c}\|_2$的值。设计数据结构S的目的是为了把S中的内容传给polyval函数做进一步处理。

函数polyval用来对输入向量决定的多项式求值，当多项式的系数由polyfit计算出来后，就可用此函数来求拟合函数。它可由以下两种方法调用：

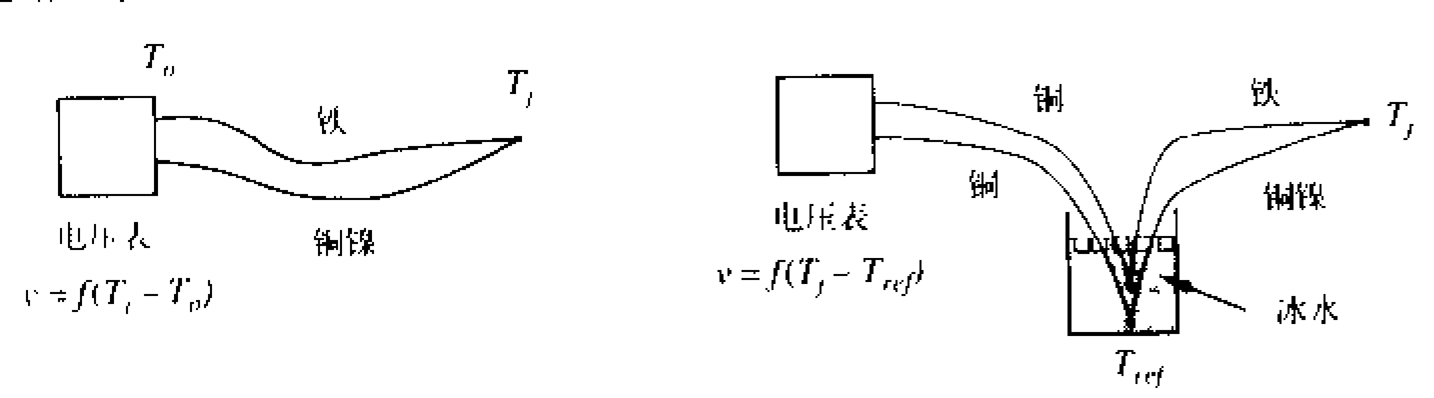
```
yf = polyval(p,xf)
[yf,dy] = polyval(p,xf,S)
```

其中输入参数p为polyfit函数返回的多项式的系数；xf为用于多项式求值的x值。输出为yf，它是由p所定义的多项式在xf上计算的值。若xf是向量，那么yf就是相同结构的向量。可选输入S是polyfit函数返回的数据结构。若S为polyval的输入，那么dy就是估计yf不确定性的向量。

函数polyfit和polyval将在下例中进行演示。

例9.12 热电偶数据的二次曲线拟合

热电偶是用来测量温度的简单装置。不同金属制成的两根导线在一端相交，另一端是开路的。由于塞贝克（Seebeck）效应，当有温度差时，会在开路和导线接头之间产生一个很小的直流电压（在温度测量文献资料中常称为emf(电动势)）。下面的简图就描述了两种常用的热电偶电路。本例中我们用J型热电偶，它由铁和铜镍合金两种导线构成。



496

左边的电路中，一只电压表（一种阻抗为无穷大的设备）用来测量开路电压。电压表的读数电压v与温差T_j-T_0有关，其中T_0是开路温度（即电压表两端的温度），T_j是铁和铜镍接点处的温度，右边的简图表示一个用冰点作为参考点的热电偶电路。因为冰的熔点已知，所以T_j的值在测出热电偶的T_j-T_{ref}的值之后就可以确定。通常用冰点作为参考来给出热电偶的校准（标定）值。

程序清单9-7中的demoTcouple函数使用内置polyfit函数来将J型热电偶数据拟合成一阶、二阶以及三阶多项式。$-50°C<T<250°C$范围的校准数据在NMM工具箱的Jtcouple.dat文件中。函数demoTcouple用来进行拟合，并画出拟合曲线与原始数据的比较图和残差的变化图

$$r_i = T_i - T_{fit,i}$$

其中T_i是封装在文件Jtcouple.dat中的数据。运行demoTcouiple可得

```
Curve fit coefficients
                constant       emf          emf^2          emf^3
linear       4.20570e-02   1.86210e+01
quadratic   -5.43114e-01   1.95312e+01  -8.29058e-02
cubic       -1.66081e-01   1.98646e+01  -1.92361e-01   6.60508e-03

Residuals
                ||r||_2      max error
linear          9.92617       4.77449
quadratic       3.18441       1.48657
cubic           0.65125       0.31147
```

程序清单9-7 函数demoTcouple使用内置polyfit函数来对热电偶校准数据进行多项式拟合

497
498

```
function demoTcouple
% demoTcouple  Linear and quadratic fits to J-type thermocouple data
%
% Synopsis:  tcouple
%
% Input:     None
%
% Output:  Print fit coefficients and residuals.  Plot fit fcns and residuals

[v,t] = loadColData('Jtcouple.dat',2,1,3);   %  Read t = f(v) data from file

% --- Perform fits, evaluate fit function, compute residuals
vfit = linspace(min(v),max(v));
c1 = polyfit(v,t,1);  tfit1 = polyval(c1,vfit);   r1 = t - polyval(c1,v);
c2 = polyfit(v,t,2);  tfit2 = polyval(c2,vfit);   r2 = t - polyval(c2,v);
c3 = polyfit(v,t,3);  tfit3 = polyval(c3,vfit);   r3 = t - polyval(c3,v);

fprintf('\nCurve fit coefficients\n                          ');
fprintf('constant        emf              emf^2            emf^3\n');
fprintf('linear    ');  fprintf('  %14.7e',fliplr(c1));  fprintf('\n');
fprintf('quadratic');   fprintf('  %14.7e',fliplr(c2));  fprintf('\n');
fprintf('cubic     ');  fprintf('  %14.7e',fliplr(c3));  fprintf('\n');

% --- Plot fit and residuals
plot(v,t,'o',vfit,tfit1,'--',vfit,tfit2,'-',vfit,tfit3,':');
legend('Data','Linear','Quadratic',2);   % Legend in upper left corner
xlabel('emf (mV)');       ylabel('Temperature ({}^\circ F)');

f = figure;   %  new figure window for residuals
plot(v,r1,'o',v,r2,'s',v,r3,'d');  legend('Linear','Quadratic','cubic');
xlabel('emf (mV)');       ylabel('Temperature residual    ({}^\circ F)');

fprintf('\nResiduals\n                  ||r||_2      max error\n');
fprintf('linear          %8.5f     %8.5f\n',norm(r1),norm(r1,inf));
fprintf('quadratic       %8.5f     %8.5f\n',norm(r2),norm(r2,inf));
fprintf('cubic           %8.5f     %8.5f\n',norm(r3),norm(r3,inf));
```

函数demoTcouiple所画图如图9-12和图9-13所示。

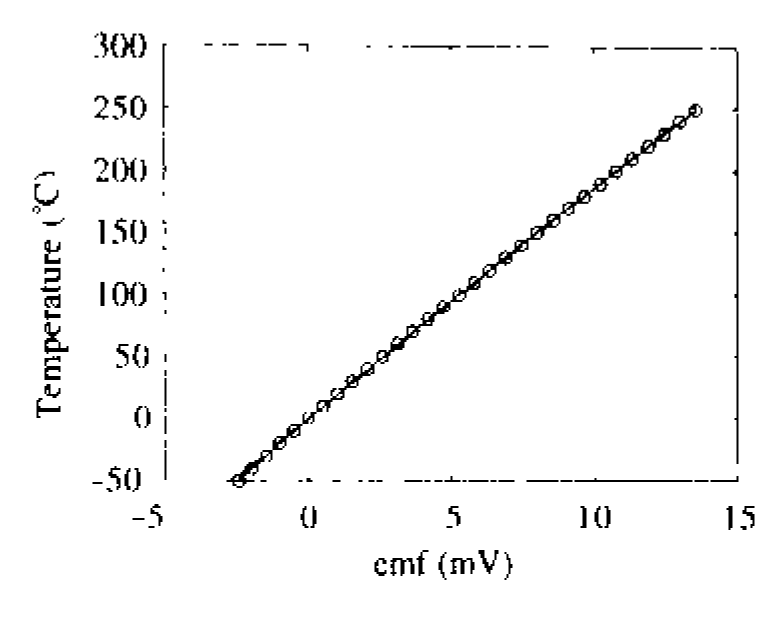


图9-12　热电偶校准数据的线性、二阶和三阶多项式拟合

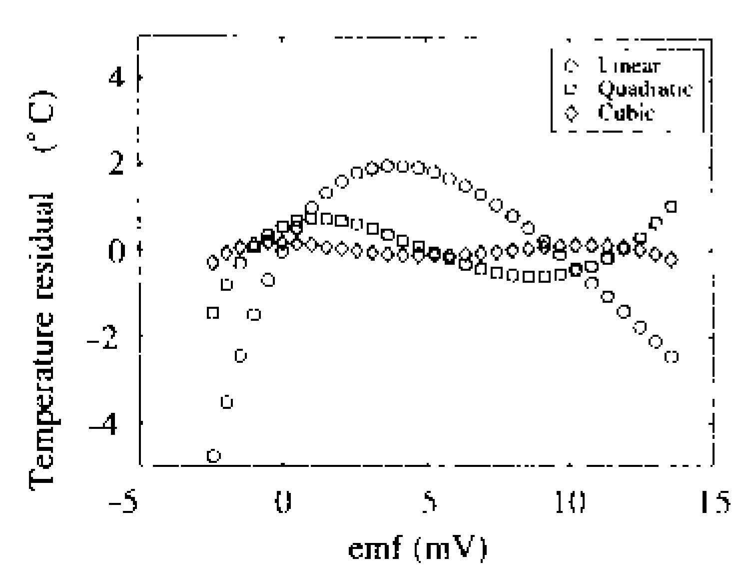


图9-13　热电偶校准数据的线性、二阶和三阶曲线拟合的残差

观察图9-12可知，直线拟合实际上和二阶和三阶拟合得同样好。打印出的 $\|r\|_2$ 和 $\|r\|_\infty$（输出为"无限大错误"）的值更精确地反映了拟合质量。增加多项式的阶会减小 $\|r\|_2$ 和 $\|r\|_\infty$。

假设拟合的目的是求将热电偶测量值转变成温度的简便公式。对直线拟合，由于拟合函数和热电偶测量值之间不一致，最大会出现4.8°C的误差；若使用二阶拟合，相应的误差为1.5°C；三阶拟合的相应误差为0.3°C。在很多热学实验中，0.3°C误差是可以接受。实际上，在由测量的emf计算T时，使用复杂一点的刻度方程会排除转化误差。

要更深一步了解拟合质量，可以研究每个原始输入数据对的残差变化。图9-13是三个拟合的$r = T_i - T_{fit}$与emf关系图。直线拟合的残差有很强的抛物线趋势（即残差与emf相关联）。添加二次项之后，大大减小了 $\|r\|_2$，因为它消除了r和emf之间关系的二次项影响。二阶拟合的残差显示了与emf的三阶变化趋势，而三阶拟合也明显地有更高阶的变化。随着拟合多项式的阶从一增到三，残差值的变化程度明显减小。

一般地，可以找到基本函数的组合，得出与自变量无关联的残差。理想数据集上没有这种关联，这说明拟合函数能解释数据中除噪声（误差）以外的所有东西；但真实数据集中可能包含着潜在的关联，仅仅是拟合基本函数集的不足，并不能也不应该用来解释这些关联。例如，一个实验中，如果对输入变量有不可控制的、可能检测不出的、瞬时的激发因素，其后数据点就会与时间有关联。

499

如果读者对残差值的检验有兴趣，可以参考回归分析方面的书。Draper和Smith[19,2.3节]就是很好的入门书。

9.3　多元线性最小二乘拟合

前面介绍的数值方法适用于将数据拟合为单个自变量的函数$y=F(x)$，其一般实现可归结为求解超定方程组的近似解。将这个框架加以拓展，就得到数据拟合为多自变量函数的问题，即$y = F(x_1,x_2,\ldots,)$。

最小二乘法在数据分析中的一个一般应用就是多重线性回归（*multiple linear regression*），其中假设因变量y线性关联于p个独立自变量

$$y = c_0+c_1x_1+c_2x_2+\cdots+c_px_p$$

在与$y=f(x)$对应但更简便的例子中，我们讨论包含数据对(x_i,y_i)的数据，$i=1,\ldots,m$。一般地，在多元拟合中我们讨论元组数据集合$(x_{i1},x_{i2},\ldots,x_{ip},y_i)$，$i=1,\ldots,m$。可以设想一个实验，其中参数$x_1,\ldots,x_p$确定，结果$y$可以测量得到，将此实验重复$m$次，就得到$m$个数据元组。

最小二乘法并不仅仅限于p个参数$x_1,\ldots,x_p$为线性的函数。一般地，多元最小二乘拟合（*multivariate least-squares fit*）具有如下形式

$$y=c_1f_1(x_1,x_2,\ldots,x_p)+c_2f_2(x_1,x_2,\ldots,x_p)+\cdots+c_nf_n(x_1,x_2,\ldots,x_p) \tag{9-42}$$

其中$f_1,\ldots,f_n$是拟合的基本函数。例如，拟合函数

$$y=c_1x_1+c_2x_2+c_3x_1x_2$$

具有基本函数$f_1=x_1$，$f_2=x_2$，$f_3=x_1x_2$。

MATLAB中多元拟合是将单个变量情况作一点小小的改动而实现的，前后惟一的差异就是描述超定方程组的矩阵形式不同。为了详细说明拟合问题，我们在p个自变量中需要一个有m个数据元组的集合

$$\begin{matrix} x_{11} & x_{12} & \cdots & x_{1p} & y_1 \\ x_{21} & x_{22} & \cdots & x_{2p} & y_2 \\ \vdots & \vdots & & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{m,p} & y_m \end{matrix}$$

和一个有n个基本函数$f_1,\ldots,f_n$的集合。方程（9-42）中的拟合系数c_j就是下式的最小二乘解

500

$$\boldsymbol{Ac}=\boldsymbol{y} \tag{9-43}$$

其中

$$\boldsymbol{A}=\begin{bmatrix} f_1(x_{11},\ldots,x_{1p}) & f_2(x_{1,1},\ldots,x_{1,p}) & \cdots & f_n(x_{11},\ldots,x_{1p}) \\ f_1(x_{21},\ldots,x_{2p}) & f_2(x_{21},\ldots,x_{2p}) & \cdots & f_n(x_{2,1},\ldots,x_{2p}) \\ \vdots & \vdots & & \vdots \\ f_1(x_{m1},\ldots,x_{mp}) & f_2(x_{m1},\ldots,x_{m,p}) & \cdots & f_n(x_{m1},\ldots,x_{m,p}) \end{bmatrix}$$

$\boldsymbol{y}$是由因变量的值形成的向量。

在单个变量函数的最小二乘拟合中，所有不确定因素都假定在因变量上。故而，若方程（9-42）描述了实验的结果，变量$x_1,\ldots,x_p$就可假设已精确地给出。最小二乘拟合将拟合函数（拟合表面）和因变量之间距离的L_2范数最小化。结果就是集合c_j，$j=1,\ldots,n$，它们满足

$$\min\|\boldsymbol{y}-\boldsymbol{Ac}\|_2$$

例9.13 数据的平面拟合

方程

$$y=c_1x_1+c_2x_2+c_3$$

表示在三维空间的一个平面。为简明起见，我们将变量重命名为$x_1\to x$，$x_2\to y$，$y\to z$，得

$$z=c_1x+c_2y+c_3 \tag{9-44}$$

此公式中，自变量是x和y，基本函数是$f_1=x$，$f_2=y$，$f_3=1$。超定方程组的矩阵为

$$A=\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_m & y_m & 1 \end{bmatrix}$$

在最小二乘问题中使用此矩阵时，假设拟合函数会接近但不会经过输入值z。另一种形式（还有其他形式）是

$$a_1x+a_2y+a_3z=1 \tag{9-45}$$

这里，假设变量x，y，z已知，且拟合函数会在逼近右边常数的1过程中将误差最小化。 501

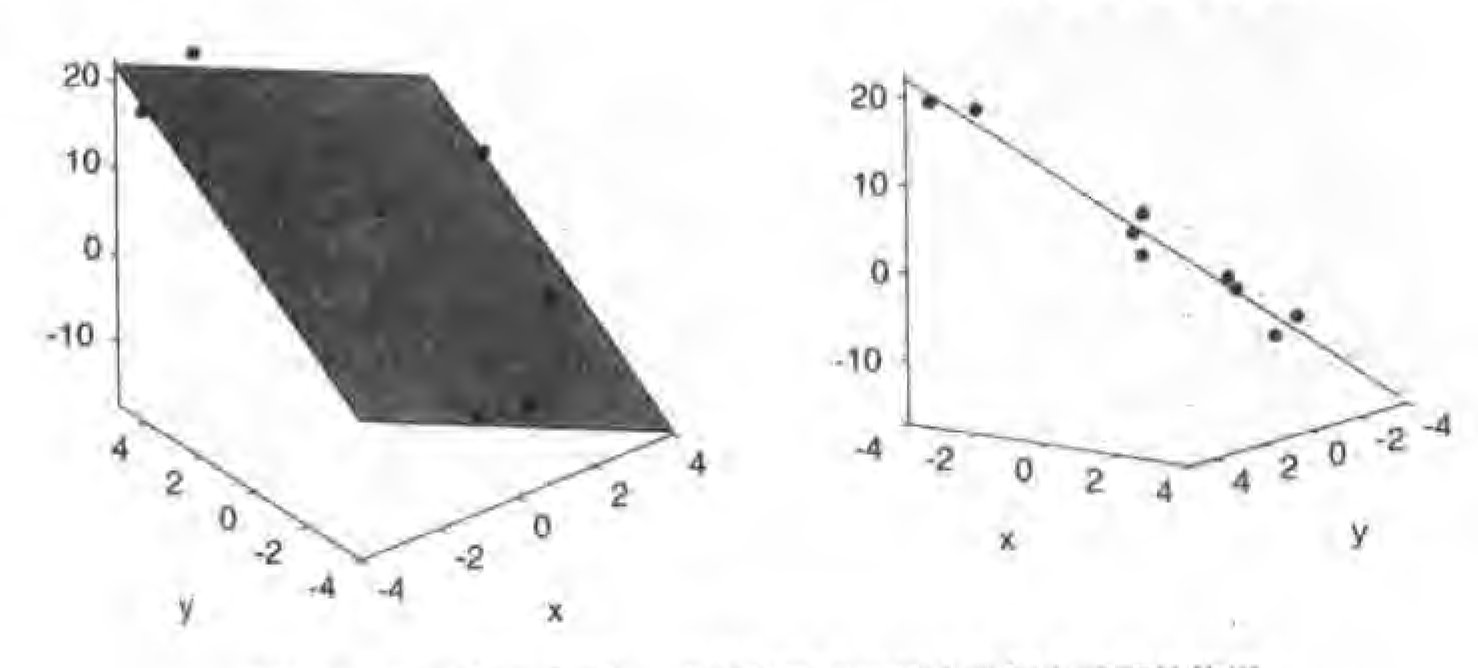
在最小二乘的意义上，思考一下如何找到与下列数据最接近的平面。

x	−4	−3	−2	−1	0	1	2	3	4
y	4	−3	5	−4	1	−3	4	−1	3
z	18.74	−1.10	19.88	−5.71	6.20	−10.37	4.96	−5.30	1.54

此数据是随意地选择一些x和y的值，并通过公式$z=-2x+3x+1$+noise（噪声）计算出了z。噪声使数据并不精确地在一个平面上。将此数据进行方程（9-44)的最小二乘曲线拟合，可通过下面的MATLAB语句实现

```
>> x = [-4     -3    -2      -1     0      1      2      3      4  ]';
>> y = [ 4     -3     5      -4     1     -3      4     -1      3  ]';
>> z = [18.74  -1.10  19.88  -5.71  6.20  -10.37  4.96  -5.30   1.54]';
>> A = [x   y   ones(size(x))];
>> c = A\z
c =
   -1.9742
    2.5543
    1.5016
```

如果没有噪声加到z上，可轻易证明原始系数c就是最小二乘解的返回值。



502

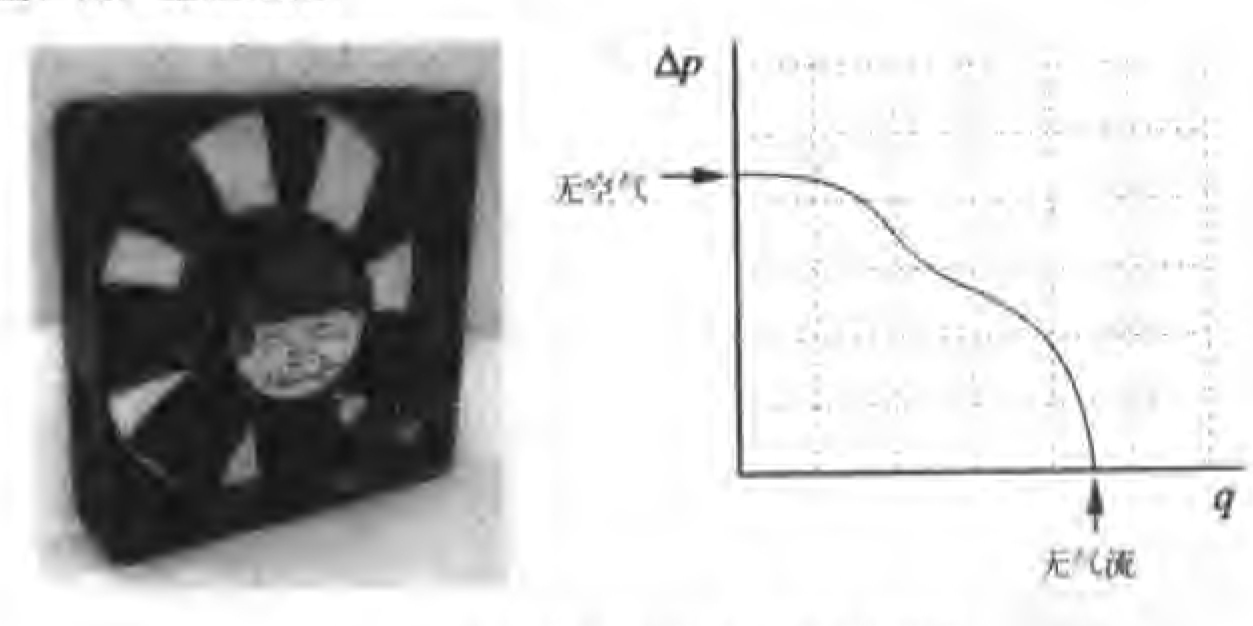
图9-14 数据的平面拟合。左边的图表示拟合数据和平面是使用的MATLAB缺省视角，右边的图表示相同数据的平面视角

图9-14表示数据的两个图形以及最小二乘拟合得到的平面。左边的图将数据点描述成实心点，平面描述成阴影表面；右边的图表示相同数据放到拟合平面上来观察。由这个视角，只有平面的侧面（在视觉上是一条直线）可见，数据明显地分散在平面的上下。图中的图形由NMM工具箱中的demoPlaneFit函数产生。

例9.14　风扇曲线拟合

图9-15描述了一个常用来给电子设备降温的小型叶片轴式风扇。当设备要安装在家里或办公室里的时候，一定要保证风扇足以降温，而且要尽量地安静。这些目标是矛盾的，因为噪声和风速都会随着转速的提高而增加。让这些直流风扇减少噪声的一个普遍的方法是选择一个更大的风扇，并让它在低于额定电压的电压下工作。降低电压可使风扇速度降低，因此可以减少空气动力噪声，而由于风扇很大，让它在低压下工作就可以提供同样的风流量，却可以产生比满负荷工作的小风扇更小的噪声。

为了选择出适于这种应用的风扇，工程师需要知道风扇的风流量和电压的关系，以及它所在系统的流体阻抗。依据某种具体风扇的特性可以画出它的"风扇曲线"，如图9-15中右图所示。风扇曲线可以通过测量风扇工作时要克服的压强差Δp以及由此产生的气流量q得到。风扇曲线的一端，所谓无空气条件(free-air condition)，相当于风扇在无压强差的自由状态下工作，可以设想一下没有任何气流障碍的风扇。风扇曲线的另一端是无气流条件(no-flow condition)，相当于风扇在出风口完全阻塞的情况下工作。虽然没有气流，但是风扇仍然在扇叶和最近的出口间产生压强差。



503

图9-15　翼式轴流风扇以及典型风扇曲线

直流风扇的制造商通常只给出设计电压（一般是12伏）下的风扇曲线。工程师要想在其他电压下运行风扇，就需要额外的风扇曲线，通常它是电压的函数。换言之，工程师需要一组$\Delta p=F(q,v)$曲线来代替制造商提供的$\Delta p=F(q)$曲线，其中v是拟提供给风扇的工作电压。图9-16描述了一种特殊风扇测得的实验数据，其中的记号表示实验数据，实线由最小二乘拟合获得。图9-16的数据包含在NMM工具箱的data目录下的文件序列fan7v.dat，fan8v.dat，...，fan13v.dat中，由文件名可知道每个文件关联一种单独电压。注意不同电压的文件有不同数目的数据点。

图9-16中的实线由对下面方程的拟合得到。

$$\Delta p=c_1+c_2q+c_3q^2+c_4q^3+c_5v+c_6vq^2+c_7vq^3 \qquad (9\text{-}46)$$

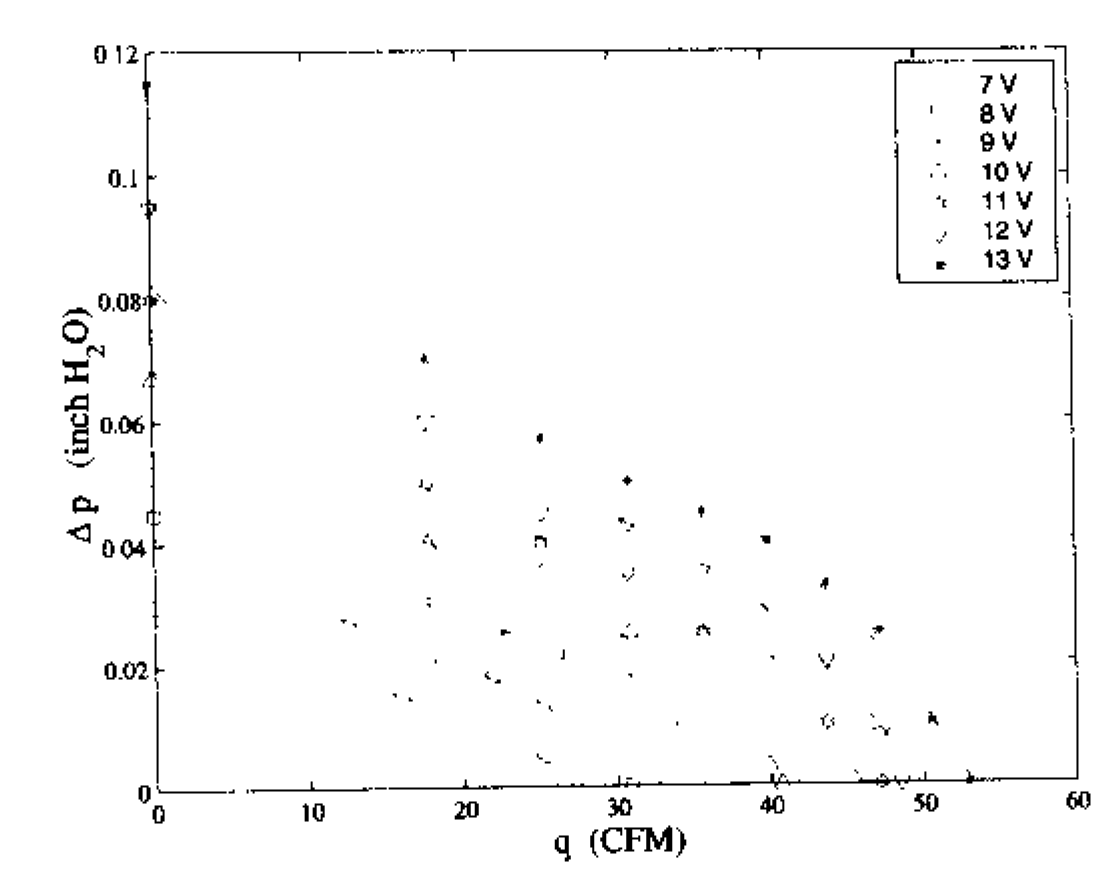


图9-16 风扇曲线数据在电压范围上的表面拟合

此方程相当于v的一阶和q的三阶多项式的乘积，即

$$\Delta p = (a_1 + a_2 v)(b_1 + b_2 q + b_3 q^2 + b_4 q^3)$$

程序清单9-8中的demoFanCurve函数从文本文件中读出$\Delta p = F(q,v)$数据，并对方程（9-46）进行最小二乘拟合。函数demoFanCurve有三个主要部分：（1）读出数据并画图，（2）以公式表示并求解最小二乘问题，（3）在输入电压数据上解出曲线拟合函数并画图。

函数loadColData将每个电压的$\Delta p = F(q)$数据集读入MATLAB变量中，此函数在NMM工具箱的utils目录下，它使用文本标题和文本列标签来读入数据文件。 504

程序清单9-8 函数fitFanCurve求解出风扇数据按方程（9-46）的多元最小二乘拟合。子函数addQVfit和amatrix13的代码在程序清单9-9中

```
function demoFanCurve
% demoFanCurve  Multivariate fit of fan data:  dp = f(q,v).  The fit function is
%               linear in v and cubic in q, including cross products
%
% Synopsis:  demoFanCurve
%
% Input:     None, all data is read from files
%
% Output:    Print out of surface fit coefficients, and plot of data and fit

% --- Load and plot experimental data
[q7,dp7]   = loadColData('fan7v.dat',2,1);
[q8,dp8]   = loadColData('fan8v.dat',2,1);
[q9,dp9]   = loadColData('fan9v.dat',2,1);
[q10,dp10] = loadColData('fan10v.dat',2,1);
[q11,dp11] = loadColData('fan11v.dat',2,1);
[q12,dp12] = loadColData('fan12v.dat',2,1);
[q13,dp13] = loadColData('fan13v.dat',2,1);

plot(q7,dp7,'o',q8,dp8,'s',q9,dp9,'+',q10,dp10,'^',...
```

```
    q11,dp11,'h',q12,dp12,'v',q13,dp13,'*')
legend('7 V','8 V','9 V','10 V','11 V','12 V','13 V');

% --- Construct global fit function:  dp = f(q,v)
q  = [q7; q8; q9; q10; q11; q12; q13];                  %  q column vector
dp = [ dp7;  dp8;  dp9;  dp10;  dp11;  dp12;  dp13]; %  dp column vector
v  = [ 7*ones(size(dp7));   8*ones(size(dp8));        %  v column vector with
       9*ones(size(dp9));  10*ones(size(dp10));       %  same length as q and dp
      11*ones(size(dp11)); 12*ones(size(dp12)); 13*ones(size(dp13)) ];

A = amatrix13(q,v);       %  Assemble matrix for overdetermined system
c = A\dp;                 %  Solve overdetermined system
fprintf('\nc = \n');
fprintf('  %14.4e\n',c); %  Print in scientific notation, one c(i) per line

% --- Evaluate curve fit at each voltage and add plot to current figure
hold on
addQVfit('amatrix13',q7,7,c);       %  add 7V curve fit to current figure
addQVfit('amatrix13',q8,8,c);       %  add 8V curve fit, etc
addQVfit('amatrix13',q9,9,c);
addQVfit('amatrix13',q10,10,c);
addQVfit('amatrix13',q11,11,c);
addQVfit('amatrix13',q12,12,c);
addQVfit('amatrix13',q13,13,c);
hold off;  axis([0 60 0 0.12]);
xlabel('q  (CFM)');  ylabel('\Delta p   (inch H_20)');
```

505

方程（9-46)的基本函数是

$$1,\ q,\ q^2,\ q^3,\ v,\ vq^2,\ vq^3$$

超定方程组矩阵的列通过计算每个数据元组上的基本函数得到。将v和q数据存储成相同长度的列向量，矩阵A可以用一行MATLAB代码来表示，即

```
A = [ ones(size(q))  q  q.^2  q.^3  v  v.*q  v.*q.^2  v.*q.^3 ];
```

程序清单9-9 函数demoFanCurve的addQVfit子函数和amatrix13子函数

```
% ==========================================
function addQVfit(Afun,q,v,a)
% addQVfit  Evaluate and plot curve fit for dp = fcn(q,v) given coefficients
%           of the global polynomial fit
%
% Synopsis:  addQVfit(Afun,q,v,a)
%
% Input:  Afun = (string) name of function to evaluate matrix of
%                the overdetermined system defined by basis functions
%            q = vector of flow rate values
%            v = scalar value of fan voltage
%            a = vector of coefficients obtained from least squares fit
%                of dp = f(q,v).  The a vector must be compatible with
%                the matrix returned by Afun
%
% Output: Plot of dp = f(q,v) is added to current figure window
qfit = linspace(min(q),1.1*max(q))';  %  Note that qfit is a column vector
vfit = v*ones(size(qfit));            %  Create v vector compatible with qfit
```

```
Afit = feval(Afun,qfit,vfit);         % Eval basis fcns at qfit and vfit
dpfit = Afit*a;                       % dp value obtained from fit
plot(qfit,dpfit)                      % Add line plot to current figure

% ========================================
function A = amatrix13(q,v)
% amatrix13  Evaluate matrix of overdetermined system for fit that
%            is linear in v and cubic in q
%
% Synopsis:  A = amatrix13(q,v)
%
% Input: q = vector of flow rate values
%        v = vector of fan voltages.  Note length(q) must equal length(v)
%
% Output:  A = matrix of overdetermined system defined by basis functions
A = [ones(size(q))  q  q.^2  q.^3  v  v.*q  v.*q.^2  v.*q.^3];
```

506

因为每个数据文件中的电压值是标量，所以必须复制电压值来组成向量v。向量v，q和dp的前几条数据有如下形式:

文件名	v	q	dp
	7	0	0.03
	7	16.0	0.015
fan7v.dat	7	22.6	0.01
	7	25.2	0.005
	7	26.5	0
	8	0	0.045
	8	12.6	0.027
	8	17.8	0.02
fan8v.dat	8	21.9	0.018
	8	25.2	0.013
	8	30.9	0
	⋮	⋮	⋮

给定v和q，矩阵A可以从amatrix13子函数中解出（见程序清单9-9）。矩阵A由基本函数定义，它们在构造最小二乘问题和计算拟合的时候都要用到。如果将A的计算放到单独的子函数中，那么我们只需定义一次基本函数，这样就简化了比较不同基本函数集合的任务（参照习题39）

程序清单9-9中的addQVfit子函数计算含q个值的向量和标量电压的曲线拟合。计算给定电压上原始数据的拟合，画出原始数据点图及拟合函数的曲线，可以清楚地看到拟合的质量。运行demoFanCurve函数可得到如下运行结果和图9-16:

```
>> fitFanCurve

c =
   -6.9398e-02
    2.9203e-03
   -1.0478e-04
```

```
 1.6968e-07
 1.3831e-02
-5.3644e-04
 1.7134e-05
-1.3002e-07
```

图9-16中的实线说明最小二乘拟合能很好地跟随离散数据的趋势，无气流条件（$q = 0$）和无空气条件（$\Delta p = 0$）下二者也比较一致。使用其他的基本函数集合可以得到稍好一点的结果（参照习题39）。要更进一步改进就需要更复杂精细的工具，如加权最小二乘和带相等性约束的最小二乘。[507]

可以将以上多元拟合与每个电压上形如$\Delta p = F(q)$的单变量拟合进行比较，这是一个很好的练习（参照习题38）。

9.4　小结

本章介绍了对已知数据曲线拟合为一组线性函数的方法。给定m个数据对(x_i, y_i)和n个基本函数$f_j(x)$，最小二乘法就是用来求解使$\|F(x) - y\|_2$最小的系数c_j，其中$F(x) = c_1 f_1(x) + \cdots + c_n f_n(x)$称为拟合函数[⊖]。对每个给定数据对写出$F(x_i) = y_i$，我们就得到了有$n$个未知$c_j$和$m$个方程的超定方程组。利用微积分来求$\|F(x) - y\|_2$，可以得到有$n$个正规方程的方程组，它可以由高斯消去法及其变化形式来求解。

从数据的直线拟合这种简单情况中，我们可以建立基本的最小二乘拟合概念。一些有两个未知c_j的非线性$F(x)$可以转化成直线方程，这称为线性变换，它的应用以$y = c_1\exp(c_2 x)$和$y = c_1 x\exp(c_2 x)$两个例子进行了示范。R^2统计量，也称为决定系数，用来对直线拟合和其他拟合函数进行质量检查。

然后将最小二乘曲线拟合过程应用于任意线性组合的函数中。这些一般拟合函数的c_j的求解方法可由数据的最小二乘直线拟合过程直接扩展得到。矩阵$\boldsymbol{A}$的列向量通过计算已知x_i的基本函数求得。c_j的超定方程组是$\boldsymbol{Ac} = \boldsymbol{y}$，其中$\boldsymbol{y}$是给定的$y$数据，向量$\boldsymbol{c}$由求解正规方程组$(\boldsymbol{A}^T\boldsymbol{A})\boldsymbol{c} = \boldsymbol{A}^T\boldsymbol{y}$决定。

除了使用正规方程组，也可以先求$\boldsymbol{A}$的QR分解，再由$\boldsymbol{c} = \boldsymbol{R}^{-1}\boldsymbol{Q}^T\boldsymbol{y}$计算出拟合系数。本章中描述了$\boldsymbol{Q}$和$\boldsymbol{R}$的属性，并开发了一些程序，用$\boldsymbol{Q}$和$\boldsymbol{R}$求解出令$\|\boldsymbol{Ac} - \boldsymbol{y}\|_2$最小的$\boldsymbol{c}$。这里没有介绍计算矩阵的QR分解的算法，而是使用MATLAB中的\ 操作符，它能在$\boldsymbol{A}$的行多于列的情况下由表达式$\boldsymbol{c} = \boldsymbol{A}\backslash\boldsymbol{y}$自动地求解出$\boldsymbol{Ac} = \boldsymbol{y}$的最小二乘解。

本章还演示了内置函数polyfit和polyval在寻找和求解多项式最小二乘拟合中的应用。函数polyfit用来具体构建超定方程组并求解出最小二乘解，然后由函数polyval计算多项式。函数polyfit是任意线性组合函数拟合的一种特殊情况。[508]

本章的最后部分描述了多元曲线拟合，其中，因变量y是p个自变量x_1，x_2，…，x_p的函数。多元拟合新出现的特征是每个基本函数都可以依赖于所有p个自变量，而不是像单变量拟合那样只依赖于变量x。本章给出了两个多元拟合的例子。

有三个m文件函数linefit、fitnorm和fitqr包含了本章中的基本数值方法。其中linefit用来获得数据对直线的最小二乘拟合，fitnorm和fitqr用来求(x, y)数据对任意线性组合的函数的最小二乘拟合。fitnorm通过求解正规方程组来求解超定方程组，fitqr

---

⊖　$F(x)$是求解x处的拟合函数时得到的列向量，其中x和y是已知数据的列向量。

则直接使用QR分解来解超定方程组。要用`fitnorm`和`fitqr`函数来求最小二乘拟合，用户必须提供一个m文件，从拟合的基本函数中求出A。

表9-2列出了本章开发的m文件函数。

表9-2　本章介绍的曲线拟合函数。小节号中带N.A.（没有应用）的没有在文章中列出，但包含在NMM工具箱中

函　数	小　节	描　述
`conductFit`	9.2.2	低温下铜的导热性数据的拟合
`cuconBasis1`	N.A.	为形如$\gamma(T)=c_1/T+c_2T^2$的拟合计算基本函数
`cuconBasis2`	N.A.	为形如$\gamma(T)=c_1/T+c_2T+c_3T^2$的拟合计算基本函数
`demoFanCurve`	9.3	风扇数据对方程（9-46)的多元最小二乘拟合
`demoPlaneFit`	N.A.	合成数据对平面方程$z=c_1x+c_2y+c_3$的拟合
`demoTcouple`	9.2.4	用`polyfit`函数对热电偶校准数据进行线性和二次拟合
`demoXexp`	9.1.5	合成数据对方程$y=c_1x\exp(c^2x)$的拟合演示
`fitnorm`	9.2.2	通过解正规方程组求最小二乘拟合
`fitqr`	9.2.3	通过QR法解超定方程组求最小二乘拟合
`linefit`	9.1.3	数据对$y=ax+b$的最小二乘拟合
`loadColData`	N.A.	从一个包含文本和数字的文件中读出数值数据，此文件可包含文本头和文本列标签。见NMM工具箱中的`utils`目录
`xexpfit`	9.1.5	数据对$y=c_1x\exp(c_2x)$的最小二乘拟合

补充读物

数据的最小二乘拟合问题一般从数值或统计的角度来讨论。在数值角度上，也就是这里采用的角度，着重于用正规方程法、QR分解法和SVD法来求出超定方程组的解。而统计角度将最小二乘作为一种回归分析的工具，以从数据集中提取出关键信息为目的。虽然读者需要在两种不同的符号集合之间进行切换，但是这两种角度是一致的。

509

在关于数值方法的书中，Gill等[28]和Kahaner等[43]对正规方程法和QR分解法都有较好的论述。Lawson和Hanson [49]（1995年改版）对最小二乘问题的数值算法研究得较早，也是这方面重要的资料。对最小二乘问题的数值分析有兴趣的读者可以参考Björk [7]。

最小二乘问题的统计算法作为例子在[19、37、52、63]中都有涉及。Draper和Smith[19]是这方面的经典著作，其中包含了很多实际的建议和例子。Hocking[37]从数学角度简单易懂地讨论了回归分析和方差分析。Mason等人在[52]中从一个实验设计和数据归纳者的角度描述了曲线拟合和统计分析。Ryan [63]提供了一些标准和最近发展形势的信息。[19、37、52、63]这些书包含内容远多于本章的主题。

Ayyub和McCuen[5]从土木工程学和环境科学应用的角度讨论了最小二乘的统计方法，Bevington和Robinson[6]从物理应用角度讨论了类似方法，并给出了非线性最小二乘问题的解决方法。

Mathworks公司还出售统计工具箱（*Statistics Toolbox*）软件包，其中包含了很多线性和非线性曲线拟合的m文件函数。类似的还有Anders Holtsberg[⊖]的*Stixbox*和Gordon K.Smyth[⊖]

⊖ www.maths.lth.se/matstat/stixbox/

⊖ www.maths.uq.edu.au/~gks/matlab/statbox.html

的*Statbox*，但它们都是免费的。卡耐基梅隆（Carnegie Mellon）大学的统计系保存了统计分析方面的大型代码库，网址为http://lib.stat.cmu.edu/。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

很多练习要使用存储在NMM工具箱的data目录下的数据。数据文件和练习的对应关系如下表所示：

510

文件	练习	描述
airSat.dat	33	不同温度下空气的饱和压强数据
airSoundSpeed.dat	30	空气中声音的传播速度和温度数据
airVisc.dat	14	空气粘度和温度数据
bearing.dat	3	不同温度下轴承磨损的数据
cucon1.dat, ...,		
cucon4.dat	35	低温下三种铜合金的导热性数据
emission.dat	37	不同湿度和大气压下内燃机释放的一氧化氮数据
fan8v.dat, ...,		
fan13v.dat	38,39	风扇曲线数据，不同电压下工作的直流风扇的压力增长与气流量数据
flowsys1.dat		
flowsys2.dat	27	通过电子设备的不同气流量下的压力降数据
GPL100.dat, GPL102.dat		
GPL104.dat, GPL106.dat	21	一种人造油在不同温度下的粘度数据
glycerin.dat	2,20	甘油在不同温度下的密度、粘度，热容和导热性数据
H2Odensity.dat	18	液态水在不同温度下的密度数据
H2Ovisc.dat	19	液态水在不同温度下的粘度
pdxTemp.dat	32	Portland OR的月平均温度的逐年变化数据
R2.dat	13	计算R^2统计量的数据
thermis.dat	22	不同温度下的热敏电阻数据
xyline.dat	6	对直线拟合的样本数据
velocity.dat	31	惯性滑行测试中不同时刻的车辆速度数据

表中有一些数据文件只包含数字列，这可用内置的load函数将数据读入MATLAB的变量中。其他的数据文件具有文本标题来标识每一列数据，这些文件可用NMM工具箱中的loadColData函数来将数据读入MATLAB的变量中。

1. (1)用方程（9-9）来手工计算例9.3中数据所拟合直线的斜率和截距。
2. (1)将glycerin.dat文件里的数据用linefit函数拟合为甘油密度关于温度的函数，此数据文件在NMM工具箱的data目录中。在相同的坐标轴中画出数据和拟合曲线的比较图。
3. (1+)Lipson和Sheth（*Statistical Design and Analysis of Engineering Experiments*，1973，McGraw-Hill，p.385）给出了一种特定轴承的磨损作为操作温度函数的数据，这些数据存储在NMM工具箱的data目录下的bearing.dat文件中。使用linefit函数拟合轴承磨损关于温度的函数，在相同的坐标轴中画出数据和拟合曲线的比较图。由提供的数据求解出100个小时操作时间内磨损不超过8毫克的轴承温度的极限。511
4. (1)函数linefit通过解正规方程组来对数据进行最小二乘直线拟合。写出linefit

的另一种形式，求解出方程（9-9)的最小二乘拟合直线的斜率和截距，并用这个函数来拟合例9.3中的数据。

5. (2)已知方程（9-12)和方程（9-13)中的A、c和y，通过计算方程（9-15)中的乘法来得到方程（9-10)，也就是说，证明方程（9-15)和方程（9-10)是等价的。

6. *(2+)文件`xyline.dat`包含两列数字（x和y），它们可以很好地拟合成一条直线。写出一个m文件函数来完成下面的任务：

   (a) 求解出直线的斜率和截距，并将这些数据存储在向量c中，计算拟合残差的范数$r=\|y-Ac\|_2$。

   (b) 用前面得到的c_1和c_2中上下±40%的值作两个向量，即

   ```
   c1t = c(1)*linspace(0.6,1.4,20);
   c2t = c(2)*linspace(0.6,1.4,20);
   ```

   (c) 对于所有`c1t`和`c2t`的组合，作出元素与其拟合残差的范数相等的矩阵R，即

   ```
   R(i,j) = norm(y - (c1t(i)*x + c2t(j)));
   ```

   (d) 用内置函数`meshc`和`contour`作出R在z轴、`c1t`在x轴、`c2t`在y轴的表面图和轮廓图。

   (e) 解释表面图和轮廓图显示的数据是怎样与最小二乘拟合推导c的理论相一致的。

7. (2)只有一个未确定系数的最小二乘拟合的计算过程特别简单。推导方程求解出下列方程的（标量）系数c，假设x和y为已知向量数据：

   (a) $y = cx$.

   (b) $y = cx^2$.

   (c) $y = x^c$.

   对每个拟合方程，写出单行的MATLAB表达式来计算c。假设x和y是列向量（提示：写出超定方程组，然后变成正规方程组。）

8. (2)写出通过调用`linefit`函数来将数据拟合为$y=c_1e^{c_2x}$的`expFit`函数。使用下列数据检查你的函数：

x	1.0000	2.5000	4.0000	5.5000	7.0000
y	2.2851	1.4176	0.8794	0.5455	0.3384

512

9. (2)写出通过调用`linefit`函数来将数据拟合为$y=c_1x^{c_2}$的`powFit`函数。使用下列数据检查你的函数：

x	1.0000	2.5000	4.0000	5.5000	7.0000
y	2.2361	10.9329	24.6765	42.8382	65.0486

10. (2)函数$y = c_1/x + c_2$可以使用变量$w = 1/x$转换成线性关系$y = c_1 + c_2w$。写出通过调用`linefit`函数来将数据拟合为$y = c_1/x + c_2$的`invxFit`函数。使用下列数据检查你的函数：

x	1.0000	2.5000	4.0000	5.5000	7.0000
y	5.5231	3.0492	2.4308	2.1497	1.9890

11. (2) 函数$y = x/(c_1x + c_2)$可以用变量$z = 1/y$、$w = 1/x$转换成线性关系$z = c_1 + c_2w$。写出通过调用`linefit`函数来将数据拟合为$y = x/(c_1 + c_2x)$的`xlinxFit`函数。使用下列数据检查你的函数：

x	2.2500	2.5417	2.8333	3.1250	3.4167	3.7083	4.0000
y	2.8648	1.4936	1.0823	0.8842	0.7677	0.6910	0.6366

x	0.7000	1.0714	1.4429	1.8143	2.1857	2.5571	2.9286	3.3000
y	−0.1714	−0.3673	−0.8243	−3.1096	3.7463	1.4610	1.0039	0.8080

12. (2)例9.6中，通过向$y = 5xe^{-3x}$增加噪声可以产生一个合成数据集。修改demoXexp函数使其具有第二个输入参数x0。使用修改后的demoXexp函数来进行以下计算：

    (a) x0 = 0.01, n = 5,10,50,100,200

    (b) n = 20, x0 = 0.1,0.01,0.001,0.0001,0.00001

    解释此实验显露的趋势（提示：画出$(\ln y - \ln y_{fit}) - x$图会有助于解题）。

13. (2)对NMM工具箱的data目录下的R2.dat文件中的数据(x, y)进行最小二乘拟合。R^2的值是多少？将最后的(x, y)数据对去掉，重新拟合。R^2的新值是多少？画出数据点图及以上两个拟合的函数图。假设最后的数据对是一个游离点，评价拟合的质量。

14. (2+)由温度决定的气体粘度可由以下Sutherland方程表示：

$$\mu = \frac{bT^{3/2}}{T+s}$$

其中μ为粘度，T是绝对温度，b和s是经验常数。给定$\mu(T)$数据，b和s的值可由对下式进行最小二乘拟合得到：

513

$$\frac{T^{3/2}}{\mu} = \frac{T}{b} + \frac{s}{b}$$

使用NMM工具箱的data目录下的airVisc.dat文件中的数据求解出空气的b和s的值。文件airVisc.dat的第一列是用摄氏度表示的温度，第二列是用kg/(m·s)表示的粘度。在拟合之前要将温度由摄氏度转变成热力学温度。

15. (2)参照fitnorm函数，写出函数fitPlot来进行最小二乘曲线拟合，并在输入数据x上画出100个点的拟合图。函数fitPlot应该调用用户自定义的求解基本函数的程序来完成拟合。（提示：参见例9.7。）

16. (2)以算法9.1中的步骤为指导，由完全QR分解法对以下数据进行直线拟合：

x	1	2	3
y	1	3	4

（这是9.2.3节中讨论的三个数据点的数据集）。列出直接计算$|z_3| = \|r\|_2$的MATLAB语句来进行验证。

17. (2)说明A的简略QR分解中的$\tilde{R}$是A^TA的Cholesky因式。（提示：将A的简略QR分解代入A^TA。）

18. (2−)饱和液态水4℃下的密度可作为温度的函数，求解出此函数关系的多项式曲线拟合系数。拟合数据在NMM工具箱的data目录下的H2Odensity.dat文件中。在表格数据点上计算每个多项式时，拟合函数的最大误差是多少？

19. *(2) 液体粘度与其温度有很大关系，其函数关系表示为：

$$\ln\left(\frac{\mu}{\mu_0}\right) = c_1 + c_2\frac{T}{T_0} + c_3\left(\frac{T}{T_0}\right)^2$$

其中，μ为粘度，T为热力学温度，μ_0和T_0是μ和T的参考值，c_i是曲线拟合常数。利用NMM工具箱中data目录下的H2OVisc.dat文件中的数据求解出水在T_0 = 0°C的c_i值。此文件的第一列是以摄氏度表示的温度，第二列是以kg/(m · s)表示的粘度。在拟合之前要将温度由摄氏度转变成热力学温度。

20. (2)对于NMM工具箱中data目录下的glycerin.dat文件中的数据，使用习题19中的拟合函数对甘油粘度关于温度的函数进行拟合，设T_0 = 0°C。

21. (3)NMM工具箱中data目录下的GPL100.dat、GPL102.dat、GPL104.dat和GPL106.dat中包含了一系列合成油的粘度与温度值。由习题19中的拟合函数，对每种合成油进行曲线拟合，设T_0 = 0°C。在相同的坐标轴上画出所有四种油的原始数据图和拟合函数图。然后，再重复进行另外三种拟合函数：一种添加$(T/T_0)^3$项，另一个添加$(T/T_0)^3$和$(T/T_0)^4$项，第三个添加$(T/T_0)^3$、$(T/T_0)^4$和$(T/T_0)^5$项。从质（通过图）和量（计算量和内存使用量）上比较这几个拟合，你推荐哪个拟合函数？ 514

22. (3)热敏电阻是用来测量温度的对温度敏感的一种电阻。电阻和温度的关系近似于下面形式的函数：

$$T = \frac{1}{c_1 + c_2 \ln R + c_3 (\ln R)^3}$$

NMM工具箱中data目录下的thermis.dat文件包含了某种热敏电阻的校准数据。

(a) 求解出这种热敏电阻的系数c_1、c_2和c_3，其中T用°C表示（提示：使用变量的近似变换，可以拟合为$1/T$关于$\ln R$的函数）。

(b) 将温度转变成热力学温度，T(K) = T(°C) + 273.15，重复(a)中的曲线拟合。

(c) 比较(a)和(b)中拟合的残差，解释其差异。拟合的残差是$r = T - T_{fit}$，其中T_{fit}是拟合函数预测出来的温度值。

23. *(2)（摘录自P.W. Atkins, *physical chemistry*, second edition, 1982, W.H. Freeman, San Francisco, p.964, problem 27.15）温度对化学反应速度的影响通常用Arrhenius方程来表示：

$$k = A \exp(-E_a / RT),$$

其中k是反应速度，A是预指数因子(preexponential factor)，E_a是活化能量(activation energy)，R是通用气体常数，T是热力学温度。某个反应的实验数据产生如下结果：

T(K)	773.5	786	797.5	810	810	820	834
k	1.63	2.95	4.19	8.13	8.19	14.9	22.2

对此数据用最小二乘拟合求解出该反应的A和E_a，其中R = 8314J/kg/K。

24. (3+)（摘录自P.W. Atkins, *physical chemistry*, second edition, 1982, W.H. Freeman, San Francisco，第826页到第827页）少量的长聚合物（高分子）溶解在溶液中对溶液的粘度影响很大。这种溶液的粘度的一种简单模型是：

$$\mu = \mu^* [1 + \eta c + \ldots]$$

其中μ^*是纯溶剂的粘度，η是溶质的所谓固有粘度，c是溶质的质量浓度。μ和μ^*使用惯用的动态粘度单位（kg/m/s），而η则使用一种反浓度（如L/gm）单位。溶质的质量浓度变高时，前面表达式中的更高次项（用“...”表示）就变得重要了。在质量 515

浓度接近零的极限中，可忽略高次项，固有粘度可由下式求得：

$$\eta = \lim_{c \to 0}\left[\frac{(\mu/\mu^{*})-1}{c}\right] \tag{9-47}$$

下表给出了聚苯乙烯溶解在甲苯中时，μ作为c的函数数据（μ的真实数据是表格中的数据乘以10^{-4}）。

c(gm/L)	0	2	4	6	8	10
$\mu \times 10^4$(kg/m/s)	5.58	6.15	6.74	7.35	7.98	8.64

确定η的值，该值在$c = 0$处截取数据对$[(\mu/\mu^*) - 1]/c$和c函数关系的曲线拟合。在计算之前，要先决定拟合将采用的适当模型。比较至少两个不同拟合函数得到的η值。

25. (2－)假设数据由两个列向量x和y给出，写出两行MATLAB代码来求直线拟合的系数。可以参考9.2.4节中的讨论。你可以用一行MATLAB语句写出吗？

26. (3－)不调用`polyfit`、`fitNorm`和`fitQR`函数，写出一个函数来将数据拟合成$y=c_1x^3+c_2x^2+c_3x+c_4$。你的函数应该将两个列向量x和y作为输入并返回系数向量。它应该是自包含的（也即是说，它应该自己构造并求解超定方程组而不调用任何的辅助函数）。使用下列数据检查你的函数：

x	−3.0000	−1.8571	−0.7143	0.4286	1.5714	2.7143	3.8571	5.0000
y	−4.9262	4.4740	−3.3136	−2.9396	−2.6697	−1.3906	−0.8669	−3.3823

27. (2+)为选择合适的风扇来给电子仪器的机壳降温，工程师要为机壳构造一个系统曲线。系统曲线是空气流量q与风扇所提供产生气流的压强差Δp之间的关系。对大多数气流系统，系统曲线的形式为$\Delta p = cq^2$。

　　某种有两个不同进气口护栅（inlet grill）的电子设备，它的试验数据存储在NMM工具箱的`data`目录下的`flowsys1.dat`和`flowsys2.dat`中。第一列是用Pa表示的压强差，第二列是用m^3/s表示的空气流量。

　　写一个m文件函数，对文件`flowsys1.dat`和`flowsys2.dat`中的数据用`polyfit`函数进行二阶多项式曲线拟合。在$0 \le q \le q_{max}$的范围上求解出多项式的拟合函数，其中q_{max}是数据集中的最大流量。将原始数据叠加到图上，当q接近零时系统曲线会发生什么？在实际中这现实吗？

28. (2)为了修正习题27中多项式曲线拟合时出现的问题，一个办法是将点$(\Delta p,q) = (0,0)$添加到数据集中。修改习题27中设计的m文件，在数据载入MATLAB变量之后，再把数据点(0,0)添加进来，以保持原始数据文件不作改变（提示：`x=[0;x]`）。对文件`flowsys1.dat`和`flowsys2.dat`中的数据再次进行曲线拟合。$q = 0$附近的问题能完全解决吗？

29. (2+)对电子仪器的机壳中的空气湍流来说（参考习题27和28），系统曲线没有一次项或常数项，也即是说，它应该是$\Delta p = cq^2$的形式。写出最小二乘拟合问题的超定方程组，求解出$\Delta p = cq^2$中的c。给出正规方程组的解析解，根据已知Δp和q数据求解出c的简单表达式。修改习题27中设计的m文件，使它除了使用`polyfit`得到的拟合外还可以用本例的拟合。对`flowsys1.dat`和`flowsys2.dat`中的数据应用此m文件。$\Delta p = cq^2$有像$Q \to 0$那样的问题吗？添加点$(\Delta p,q) = (0,0)$是否会影响拟合（提示：考虑

516

习题7的解）？

30. (2+)Hart et al.（*Optical Measurement of the Speed in Air Over the Temperature Range 300-650K*，NASA/CR-2000-210114，ICASE Report No.2000-20，Institute for Computer Applications in Science and Engineering，NASA Langley Research Center，Hampton，VA.）发明了一种用激光诱导热力声学（laser-induced thermal acoustics，LITA）来测量空气中高频声波的新技术。NMM工具箱的data目录下的airSoundSpeed.dat文件包含了从报告中摘录出来的数据。文件airSound Speed.dat的第一列是热力学温度，第二列是用LITA测量的声速，第三列是由另一种模型预测得出的声速。

理想气体中的声速为$V=\sqrt{kRT}$，其中V为声速，k是比热容，R是气体常数，T是气体温度。使用polyfit函数将LITA数据拟合为V关于$\sqrt{T}$的一阶多项式（即直线）函数。用同样的数据再对$V=c\sqrt{T}$的形式再次拟合，其中c为拟合的独立参数。在相同坐标轴上画出原始数据和两种拟合的图形，对二者进行比较。求解出由曲线拟合系数推出的k的值。（提示：练习7的解会有助于解题。）

31. (3+)（摘录自Akai[3]）一种斜坡实验可用来估计某个车辆的气动阻力（aerodynamic drag）特性。实验由车辆在已知速度的运动开始，切断动力（变速器挂成空档），当车辆慢慢停下来时测量其速度。实验结果由NMM工具箱的data目录下的velocity.dat文件给出。对失去动力的车辆由力学平衡可得

$$\sum F = F_d + F_t$$

其中F_d为气动阻力，F_t为所有使车辆慢下来的摩擦力总和（轮胎摩擦和传输系统阻力）。假设F_t可以在另一个单独的实验中求解出，但由于缺少此信息，我们现在只好先忽略F_t。由牛顿运动定律和力的平衡得

$$ma = m\frac{\mathrm{d}v}{\mathrm{d}t} = -F_d，或\ \ F_d = -\frac{1}{m}\frac{\mathrm{d}v}{\mathrm{d}t}$$

517

通过方程可由速度数据估测出F_d。

一种计算dv/dt的方法是对原始数据的有限差分逼近：

$$\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{v(t_i)-v(t_{i-1})}{t_i - t_{i-1}}$$

另外，也可以对数据进行曲线拟合，例如拟合为三阶函数：

$$v(t) = c_1t^3+c_2t^2+c_3t+c_4$$

然后对拟合函数取微分得出$\mathrm{d}v/\mathrm{d}t$。

为前面的两种方法写出必需的m文件，计算出$\mathrm{d}v/\mathrm{d}t$作为时间的函数并画图。两种方法所得的结果是否有很大的差异？

32. *(3)在NMM工具箱的data目录下的pdxTemp.dat文件中包含了在Portland,OR机场所测得的历史月份平均温度。文件包含了四列数据。第一列是月份（1到12），第二到第四列分别是每月的历史平均最低、最高和平均温度。将平均温度拟合为函数

$$T = c_1 + c_2 \sin^2\left[\frac{(m-1)\pi}{12}\right]$$

其中m是月份。画出原始数据和拟合数据之间的比较图（提示：(1)复制一月份的数据，以使数据集的开始和结尾都是一月份，如此共有13对值。(2)作出$\theta = (m-1)\pi/12$值的向量）。

33. (3+)W.C. Reynolds（*Thermodynamic Properties in SI: Graphs, tables and computational equations for 40 substances*，1979，Department of Mechanical Engineering，Stanford University，Stanford CA）给出了下列空气的饱和压强作为温度的函数方程：

$$\ln\left(\frac{P}{P_c}\right) = c_1 x + c_2 x^2 + c_3 \ln\left(\frac{T}{T_c}\right)，其中\ x = \frac{1}{T_c} - \frac{1}{T}$$

其中P_c = 3.77Mpa是临界压强，T_c = 132.5K为临界温度。由airSat.dat文件中的数据求解出$p_{sat} = f(T_{sat})$中的c_i值，此文件存放在NMM工具箱的data目录下。求解出数据的最小二乘直线拟合的c_1、c_2、c_3的值，画出原始数据和拟合函数的图形，计算拟合残差的L_2范数，即$\|p - p_{fit}\|_2$。

（注意：基本函数独立地依赖于T和T_c的值）一种解法是复制fitNorm或fitQR函数，改变函数使T和T_c的值可传给程序来计算基本函数。另一种解法是写一个专门的函数，完成以下步骤：

518

(a) 从文件airSat.dat中载入数据。

(b) 创建超定方程组的系数矩阵。

(c) 使用反斜杠操作符求解超定方程组。

34. (3)使用内置polyfit函数创建前面练习的$p - T$数据的多项式曲线拟合的一个变体。你更倾向于使用哪一种拟合？

35. (3+)用内置的polyfit函数来对NMM工具箱的data目录下的cucon1.dat，cucon2.dat、cucon3.dat文件中的导热性数据进行多项式曲线拟合（参见例9.9）

 (a) 对每个数据集，求解出2，3，4和5阶多项式拟合函数的系数。求解出$0 < T < 50$K范围的拟合函数并画图（函数subplot允许四个曲线图出现在同一页上）。在每张图上叠加原始数据。多项式拟合的质量如何？

 (b) 复制cucon1.dat、cucon2.dat、cucon3.dat文件（重命名），编辑这些数据文件，将数据点(0,0)加到每个文件中。重新进行(a)中的曲线拟合。添加的数据对拟合有所改进吗？拟合满足条件$k(T=0) = 0$的情况如何？

36. (3+)习题35中(b)部分的结果可以通过使用polyfit函数对方程

$$\frac{T}{k(T)} = c_1 T^n + c_2 T^{n-1} + \cdots + c_n T + c_{n+1} \tag{9-48}$$

的拟合来加以改进。写一个m文件函数，将所改动文件（包括数据点(0,0)的文件）中的数据对方程（9-48)进行拟合。拟合的结果与例9.9中的结果相比如何？对方程（9-48)使用polyfit函数时哪个值会达到最小？

37. (2+)Lipson和Sheth（*Statistical Design and Analysis of Engineering Experiments*，1973，McGraw-Hill，p.385）给出了内燃机释放的一氧化氮作为湿度和大气压力的函数数据。

这些数据包含在NMM工具箱的data目录下的emission.dat文件中。求此数据的形如下列方程的多重线性回归:

$$\mathrm{NO} = c_1 + c_2 h + c_3 p$$

其中h是湿度，用每磅干燥空气中的颗粒数表示，p是大气压力，由英寸汞柱数表示。求解出c_1，c_2和c_3的值。

38. (3)例9.14中讨论了翼式轴流风扇的数据对方程（9-46)的多元拟合。分别作出每个电压下的多项式曲线拟合$\Delta p = F(q)$。将所有的多项式拟合图叠加到一张含所有原始数据的图中。对每个电压曲线使用不同阶数的多项式可能会更好。这些结果与例9.14中的多元拟合相比如何?

39. (3)例9.14中讨论了翼式轴流风扇数据对方程（9-46)的多元拟合。以demoFanCurve函数作为起点，对以下各式进行拟合:

(a) $\Delta p = c_1 + c_2 q + c_3 q^2 + c_4 q^3 + c_5 q^4 + c_6 v + c_7 vq + c_8 vq^2 + c_9 vq^3 + c_{10} vq^4$

(b) $\Delta p = c_1 + c_2 q + c_3 q^2 + c_4 v + c_5 v^2 + c_6 vq + c_7 v^2 q^2$

(c) $\Delta p = c_1 + c_2 q + c_3 q^2 + c_4 q^3 + c_5 v + c_6 vq + c_7 vq^2 + c_8 vq^3 + c_9 v^2 + c_{10} v^2 q +$
$c_{11} v^2 q^2 + c_{12} v^2 q^3$

519
520

为便于比较，不改变amatrix13子函数，而是为前面的每个拟合方程作amatrix13的一个拷贝。你可能想以A=feval(Afun,q,v)来代替demoFanCurve主函数中的调用A = amatrix13(q,v)，其中Afun是一个字符串，包含了求矩阵A的函数的名称，在修改后的demoFanCurve函数中Afun可以作为输入参数使用。对每个拟合方程（包括方程（9-46)），计算给定数据点拟合的残差$\|y - Ac\|_2$。你推荐哪个拟合方程?

第10章 插　值

插值是很多数值方法中的重要组成部分；插值多项式是构筑函数数值积分和常微分方程解的基石；插值理论是构成偏微分方程数值逼近的基本原理。在图像处理和信号处理中，对数据再次抽样来改变分辨率都需要用到插值技术。在很多领域中，经常要通过插值的办法来找出表格条目间的中间值。

本章将要讨论如何用插值的办法来逼近（近似）一个由数据表决定的函数。虽然基本思想可延用到二维和三维中去，但本章开发出来的素材是基于一维数据(即$y = f(x)$)的。图10-1描述了本章的组织结构。

用多项式建立插值函数的方法主要有两种。本章的前一部分，通过匹配离散数据集里越来越多的点，来增大多项式的阶数。插值多项式使用单项式、拉格朗日和牛顿基本插值公式来建立，所有这些基本插值公式在严格的算术意义上是等价的。然而，后面会看到，这些形式的数值特性却明显不同。即使不考虑算法，同样可以看到，产生一个升幂（阶）多项式在数值上也是有局限性的，其根本的缺陷是：为了匹配不断增加的数据点，需要增大多项式的阶数，因而会导致在指定点之间插值式的值出现不希望有的振荡。

521

本章主题：

1. 基本思想

介绍了插值的基本术语，以及插值和曲线拟合、插值和外插之间的区别。

2. 任意阶数的插值多项式

本节推导出了用任意阶数的多项式进行插值的方法，展示了三种不同的多项式基本插值公式——单项式、拉格朗日和牛顿插值，它们呈现不同的数值特性。另外，还给出一个例子来证明任意高阶多项式都可以在定义插值的支点间呈现出“摆动”。

3. 分段多项式插值

这里通过将非重叠（non-overlapping）的低阶多项式组合得到高精度的插值式（interpolant），推荐在程序中使用这些分段插值。Hermite和样条（spline）插值是邻接段间具有高阶导数连续性的分段多项式。此外，还开发出了使用Hermite插值和样条插值的程序。

图10-1 第10章的主题

在本章的第二部分，将插值问题再次公式化。这里没有在相关区域建立单一的高阶多项式，而是将区域中子区间上定义的低阶多项式组合产生插值函数，这样得到的分段插值式(*piecewise interpolant*)的总体精度比单一插值函数要高得多。本章也开发了一些程序来用分段线性函数、分段三阶Hermite多项式和三阶样条函数来进行插值。本章的结尾简单讨论了MATLAB的内置插值函数。

例10.1　目测插值

图10-2是两种常见的类似器件：速度计和手表的表面。指针连续滑过有离散记号的刻度尺，当指针在两个记号中间时，可以近似为最近的刻度值，或进行目测来得到更好的刻度值。在两个最近的值之间估计一个值是插值最本质的特征。

522

速度计的指针大约在速度标记80km/h和90km/h之间的2/10处，这说明车辆的速度大约为

82km/h。如图10-2所示，当手表的分针大约指在10和11之间距离的4/5处时，目测的估计时间是2:54。这些估计使用和基本函数严格一致的线性插值来表示就是：速度是指针角度位置的线性函数，时间是分针角度位置的线性函数。

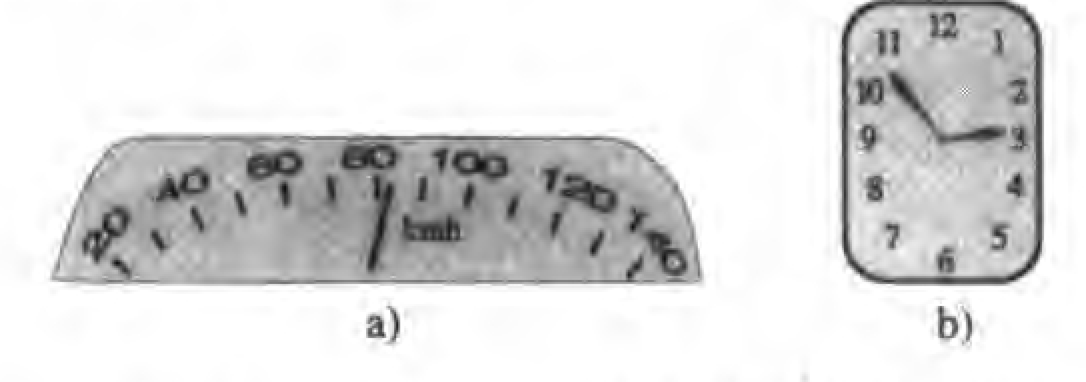


图10-2 两种常见类似器件速度计和手表的表面的目测插值

一般地，插值函数与产生插值数据的函数不同。插值函数是输出函数$y=f(x)$的逼近函数。当读出模拟表盘上的数据时，就对输入x作出了可目测估计。

例10.2 甘油粘度

甘油(丙三醇，$C_3H_5(OH)_3$)是一种液体，可作为制造从肥皂到（爆炸性的）硝化甘油(nitroglycerin)等诸多产品的原料。甘油的粘度是温度的函数，如下表[⊖]及图10-3所示：

T(°C)	0	10	20	30	40	50
μ(N · s/m)	10.60	3.810	1.492	0.629	0.2754	0.1867

此数据储存在NMM工具箱的`data`目录下的`glycerin.dat`文件中。

对表格中的数据使用线性插值，我们可以估计出甘油在22°C下的粘度为：

$$\mu(22^\circ\text{C}) \approx 1.492 + \frac{2}{10}(0.629 - 1.492) = 1.319$$

523

如例10.7中所示，使用二阶插值可得$\mu(22^\circ\text{C}) = 1.203$，误差为10%。

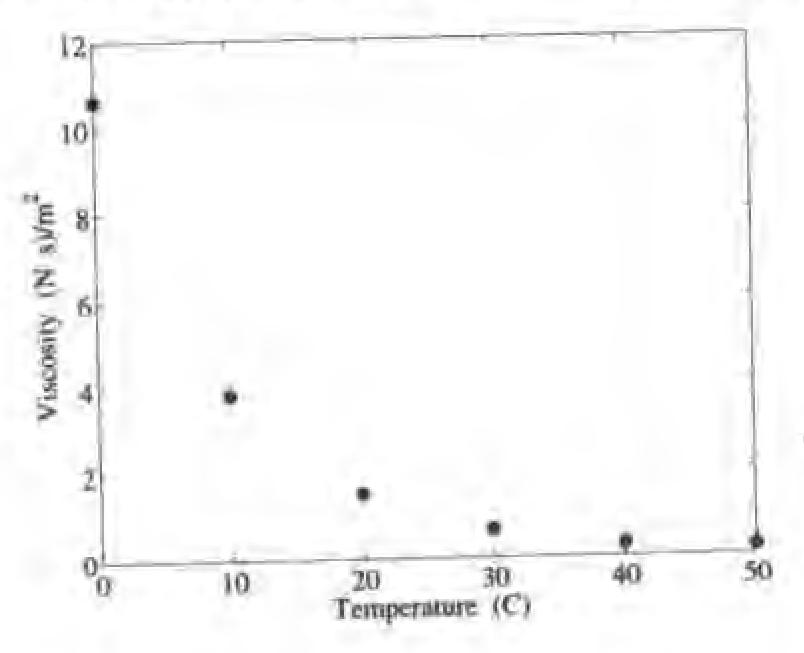


图10-3 甘油粘度与温度的函数关系

⊖ 数据摘自E.R.G. Eckert和R.M. Drake,Jr.的*Analysis of Heat and Mass Transfer*，1972，McGraw-Hill，Table B-3，p.779。

10.1 基本思想

固定数据集上的插值在数学上相当于读出数据的内在含义。一维情况下，数据集 (x_i,y_i)，$i = 1,\cdots,n$，作为一些已知或难以求解的函数$y=f(x)$的离散抽样。插值就是建立并求解出在数据集(x_i,y_i)之内或之外的值为$x = \hat{x}$的插值函数（或插值式）$y = F(x)$。插值函数$F(x)$需要通过已知数据(x_i,y_i)才能决定。对$\hat{x} \neq x_i$，$F(x)$也应该能对$f(x)$很好地逼近，其中$f(x)$首先产生表格数据。

有些情况下，$f(x)$可能以一个实验的形式存在，这样的话数据集(x_i,y_i)就表示n次测量的输入和输出值。在其他情况下，$f(x)$可能是一个已知的解析函数，但是很难求出解，特别是手工计算时。例如，很多手册中的数学函数和以表格描述的物理数据就是这种情况。不管$f(x)$是否已经分析出来，插值过程只用到数据(x_i,y_i)的有限集合。

在插值的一般形式中，它主要是求出n个基本函数$\Phi(x)$的线性组合的系数a_1、a_2、…，a_n，这些基本函数组成以下插值式：

$$F(x)=a_1\Phi_1(x)+a_2\Phi_2(x)+\cdots+a_n\Phi_n(x) \tag{10-1}$$

这样就有$F(x_i) =y_i$，其中$i = 1$、…，n。基本函数也可以为多项式

$$F(x)=a_1+a_2x+a_3x^2+\cdots+a_nx^{n-1}$$

或三角函数式

524

$$F(x)=a_1+a_2e^{ix}+a_3e^{2ix}+\cdots+a_ne^{(n-1)ix}$$

（其中$i=\sqrt{-1}$）、又或者是其他合适的函数集。多项式常用来进行插值，因为它们易于分析计算和求解。

10.1.1 插值和曲线拟合

已知通过实验或计算所得的数据集$y_i = f(x_i)$，$i = 1,\ldots,n$，当x不在原始数据集中时，往往需要通过计算的办法来求解y的值。曲线拟合是用插值办法求解这种问题的一种替代办法。在曲线拟合中，逼近函数穿过数据点附近，但通常不是精确地穿过。拟合函数与数据点不一致，这说明数据中还有不确定的因素。图10-4描述了一些假想的实验数据及其曲线拟合函数图形。

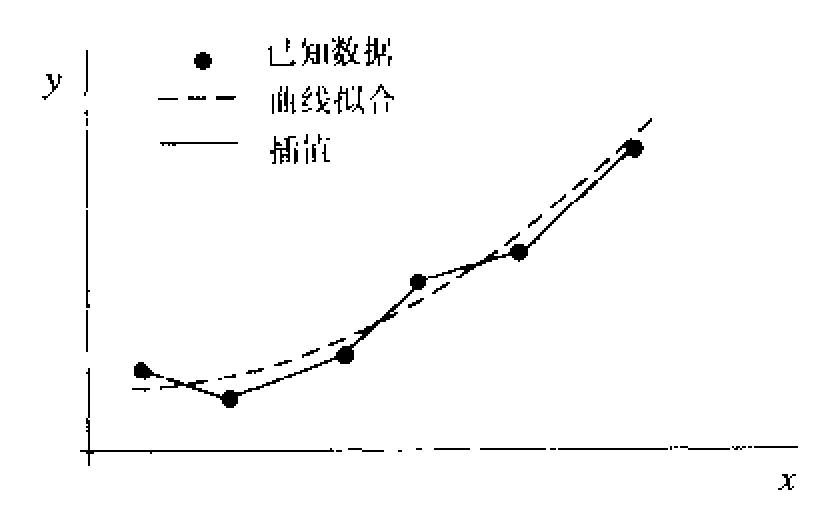


图10-4　曲线拟合函数在数据点的附近经过，而插值函数则精确地通过这些数据点

与曲线拟合相反，插值过程本身就假设数据没有不确定性。插值函数要精确地穿过每个已知数据点。图10-4也描述了假想数据使用分段线性函数进行插值的结果。曲线拟合函数的曲线是平滑的，而分段线性插值函数生成的结果则表示为由斜率不连续的线段连成的折线。当然，插值的这种不平滑性可通过使用不同的插值函数来消除（如三阶样条插值）。

虽然曲线拟合过程和插值过程有根本的差异，但是这种差异并非总是很重要的。例如，在粗略的工程计算中，对实验得出的不确定性数据点进行线性插值，可能会产生非常有效的估计数据。这种情况下，如果已经考虑到结果中的不确定性，那么就完全有理由忽略数据的这种不确定性。

10.1.2 插值和外插

插值（内插）是在已知数据集的自变量的范围内建立和计算一个逼近函数的过程。为表达简便，假设数据(x_i, y_i)的顺序为$x_1 < x_2 < \cdots < x_n$。插值提供了一种在$x_1 \leqslant x_i \leqslant \cdots \leqslant x_n$上对$y = f(x)$逼近的方法。外插(extrapolation)是在给定自变量的范围之外即$x<x_1$或$x>x_n$上求解插值函数的方法。插值和外插如图10-5所示。



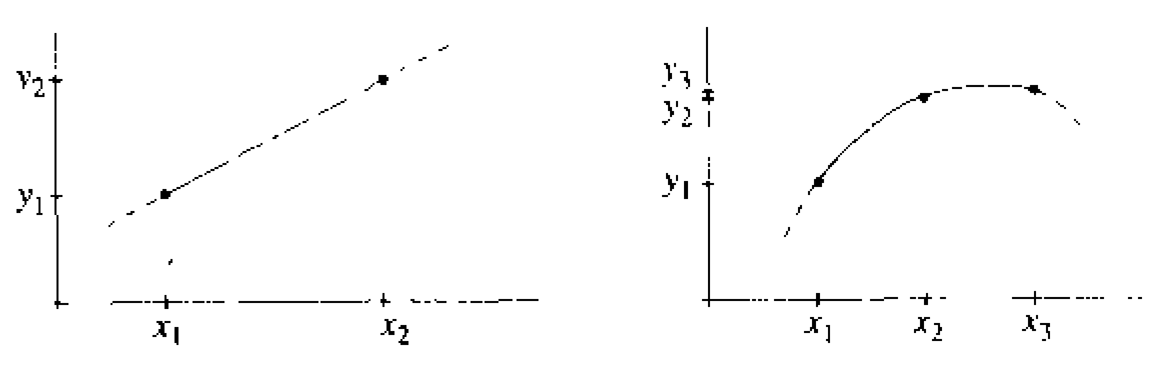


图10-5 线性和二阶多项式插值公式的插值和外插。虚线表示已知数据范围外的外插

例10.3 机场相关航运量数据的外插

繁荣的区域经济除带来了航空旅行的持续普及，也导致了20世纪90年代Portland.OR的航空公司的乘客剧增。NMM工具箱的data目录下的pdxPass.dat文件包含了从1981年到1998年途经该机场的乘客数量数据。为计划将来的航运量水平，需要估计未来五年使用机场的乘客数量。虽然数据只适用于某一段时期，但是我们仍然使用这些数据来说明使用外插可能的风险。

预测将来总是困难的。城市规划者需要使用经济模型来模拟人口和工业生产的增长，还要模拟国家和地区经济的总趋势，以此来估计机场将来的航运量状况。这里没有这种专门的模型，就只剩下历史上的乘客数据了。建立这些数据的外插函数，也许可能预测出将来的情况。将本章后来描述的插值函数进行外插，其结果如图10-6所示。所有外插函数的推断在给

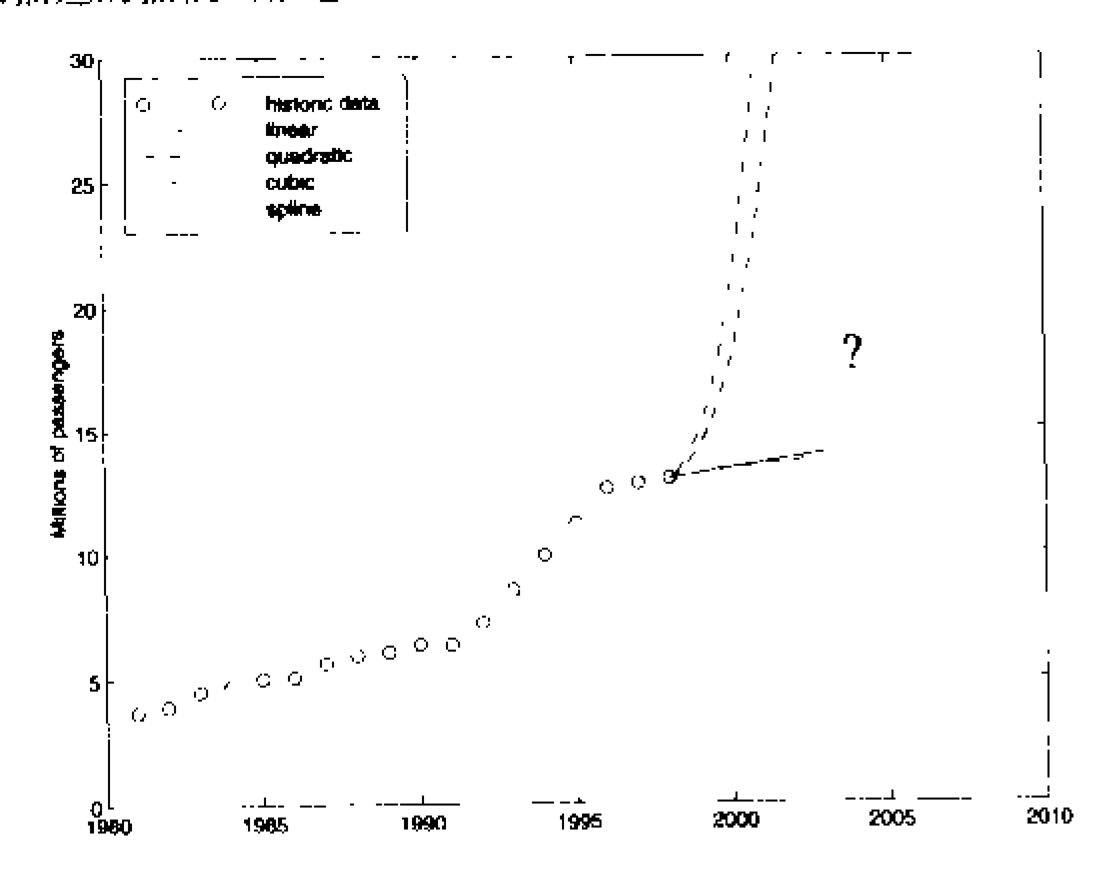


图10-6 使用本章中不同的插值函数对机场乘客数据的外插

定时间数据后面第一年的情况大约一致，但是随后它们就开始明显发散了。向后推断五年，可以得出下表所示的数值结果： 526

外插	2003年机场的客流量（百万）
线性插值	14.0
二次插值	13.8
三次插值	53.8
三次样条插值	79.5

这些预测结果不一致，就定性地说明了外插本身固有的不确定性。

10.2　任意阶的插值多项式

本节讨论了建立和求解任意阶插值多项式的一些算法。一个有n个数据点的集合可以用$n-1$阶多项式进行插值。虽然这里讨论的算法允许建立任何n阶多项式，但实际中只有低阶多项式，比如低于五阶的多项式才是实用的。随着n的增加（即使用更多的数据点来建立更高阶多项式），数据点间的插值可能会明显偏离已知且临近的数据点，这就是将在10.2.4节中讨论的多项式的摆动(polynomial wiggle)问题。

10.2.1　用单项式基本插值公式进行多项式插值

多项式插值是求通过n个已知数据对或支点（support point）的那个$n-1$阶多项式，即方程$P_{n-1}(x)$。$n-1$阶多项式的一般表示是

$$P_{n-1}(x)=a_1+a_2x+\cdots+a_nx^{n-1} \tag{10-2}$$

527

它用一个单项式基本函数(*monomial basis functions*)x^0、x^1、x^2、…、x^{n-1}的集合来定义多项式。虽然用单项式表示法很简练，读者对此也比较熟悉，但是多项式$P_{n-1}(x)$可以写成其他的形式。考虑多项式的另一种形式

$$P_{n-1}(x-\xi)=b_1+b_2(x-\xi)+\cdots+b_n(x-\xi)^{n-1} \tag{10-3}$$

其中ξ为已知偏移量。合适地定义系数b_i，方程（10-2）和方程（10-3）可定义相同的多项式。

MATLAB中计算多项式的内置程序（另见2.3.3节）需要将多项式写成x的降幂形式

$$P_{n-1}(x)=c_1x^{n-1}+c_2x^{n-2}+\cdots+c_{n-1}x+c_n \tag{10-4}$$

这里的c_i与方程（10-2）中的a_i的值来自相同的数据集，只不过是顺序不同。

Vandermonde方程组　已知由n个有序数据对(x, y)构成的集合，可以使多项式通过每个数据点，并为n个未知系数c_i写出n个方程，由所得到的方程组可解出c_i。

考虑建立二阶插值函数

$$y=c_1x^2+c_2x+c_3 \tag{10-5}$$

此函数通过支点$(-2,-2)$、$(-1,1)$和$(2,-1)$。将已知点代入方程（10-5）可得关于c_i的由三个方程所组成的方程组：

$$\begin{aligned}-2&=c_1(-2)^2+c_2(-2)+c_3\\ 1&=c_1(-1)^2+c_2(-1)+c_3\\ -1&=c_1(2)^2+c_2(2)+c_3\end{aligned}$$

以上方程组可记为

$$\begin{bmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 4 & 2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \\ -1 \end{bmatrix} \tag{10-6}$$

对三个点(x_1,y_1)、(x_2,y_2)和(x_3,y_3)的任意集合，矩阵方程为

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

528

左边的系数矩阵称为Vandermonde矩阵。

方程（10-6）中的Vandermonde矩阵可由如下的MATLAB语句来建立：

```
>> x = [-2  -1  2]';            %  transpose required, x must be a column
>> A = [x.^2  x  ones(size(x))];
```

内置函数vander可以由任意向量创建相应的Vandermonde矩阵，前面的语句可以代替为：

```
>> A = vander([-2  -1  2]);     %  vander is clever, transpose is optional
```

函数vander设计得很灵巧，可以在建立矩阵之前将输入向量转化成列向量，其输出矩阵的维数取决于输入向量的长度，学习vander函数[⊖]中的语句具有指导意义。

一旦定义了Vandermonde矩阵，求解方程(10-6)会得到插值多项式的系数：

```
>> y = [-2  1  -1]';
>> c = A\y
c =
   -0.9167
    0.2500
    2.1667
```

虽然这个过程直观易懂，但它却不是建立插值多项式最好的方法，这是因为Vandermonde方程组有可能是病态的。

例10.4 汽油价格插值

考虑对下表[⊜]（假想的）汽油价格数据（单位：美分）的插值：

年份	1986	1988	1990	1992	1994	1996
价格	133.5	132.2	138.7	141.5	137.6	144.2

已知六个数据对，一种简单的插值策略是对数据进行五阶多项式插值：

$$p=c_1y^5+c_2y^4+c_3y^3+c_4y^2+c_5y+c_6 \tag{10-7}$$

529

其中p是价格，y是年份。下面的MATLAB语句建立方程组并求解出系数c：

```
>> year = [1986   1988   1990   1992   1994   1996 ]';
>> price = [133.5  132.2  138.7  141.5  137.6  144 2]';
>> A = vander(year);     % Set up Vandermonde system
>> c = A\price;          % and solve it

Warning: Matrix is close to singular or badly scaled.
```

⊖ 在命令提示符下键入type vander来取得该函数列表。

⊜ 某些人有理由反对像汽油这样的挥发性货物中取得的粗糙数据样本。诚然，更科学的研究还要包含更多的数据，以及充足的条件来确保数据是油价的真实总体的描述。

```
        Results may be inaccurate. RCOND = 8.305426e-32

>> fprintf('%12.4e\n',c)
  3.0382e-03
 -3.0207e+01
  1.2014e+05
 -2.3890e+08
  2.3753e+11
 -9.4465e+13
```

警告信息指出了A是病态的。RCOND[⊖]是估计出的条件数的倒数，它的值很小说明矩阵的条件数非常大。下面的语句解出了插值函数并给出了图示：

```
>> y = linspace(min(year),max(year));
>> p = polyval(c,y);                    % Eval polynomial at each y
>> plot(year,price,'o',y,p,'-')
```

图10-7显示的结果进一步证实了插值函数有严重的问题。插值函数的曲线不是预料中的平滑曲线，而是不规则曲线。

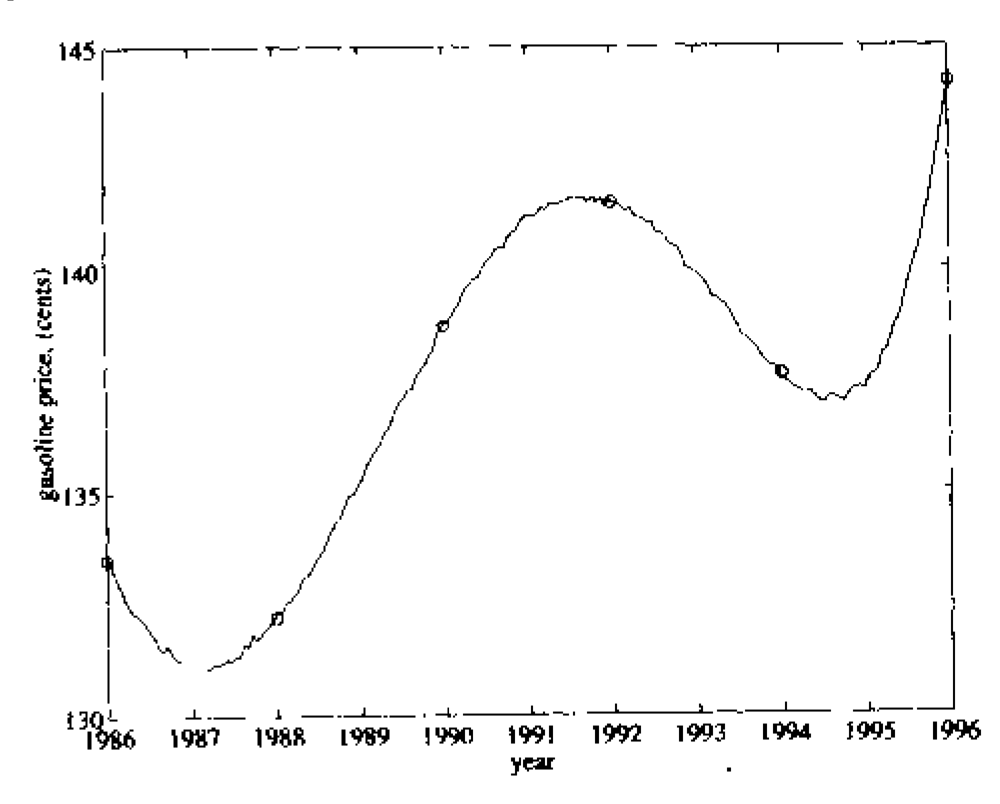


图10-7 使用单项式基本插值公式对汽油价格数据进行插值

前面的计算显露出了两个不同但是有关联的现象。首先，矩阵A是病态的。对10^{31}阶的条件数和双精度算术运算来说，方程（8-38）说明解向量c中的元素中没有正确的有效数字。矩阵A的条件数很大，因为其中的元素值的大小差异很大，它们中最小的元素是1，最大的元素是$1996^5 \approx 3.2 \times 10^{16}$。在求解过程前面的消去阶段（即在计算$L$和$U$因子阶段），很小的舍入误差会导致矩阵$L$和$U$有很大的不确定性。在向后代入过程中，会有更多的舍入误差，并对c_i的真值产生扰动。其结果为多项式

$$\tilde{p} = \tilde{c}_1 y^5 + \tilde{c}_2 y^4 + \tilde{c}_3 y^3 + \tilde{c}_4 y^2 + \tilde{c}_5 y + \tilde{c}_6 \tag{10-8}$$

它与方程(10-7)中的多项式不同。

[530] 在c_i的计算过程中，舍入误差的问题是使用单项式基本插值公式进行多项式插值时固有的，这是因为Vandermonde矩阵通常是病态的。对于消去了计算数量级在$O(10)$以上的项的低阶多

⊖ 另见内置的rcond函数

项式来说，比如$n<4$，出现这些问题的可能性会相对较小。因为高斯消去法和向后代入在方程组病态的情况下求出解的残差也可以很小，即使c_i不精确，使用单项式基本插值公式求出的多项式也可能不会有问题。

求解Vandermonde方程组时还存在第二个问题，就是插值多项式的系数在16阶的数量级中变化。使用1000阶y值求解方程(10-8)会导致很明显的舍入误差，引起图10-7中描述的高频振荡。由于多项式求解中的舍入误差与不可靠的c_i值，插值函数对输入数据的逼近会变得不精确。

如本例演示的那样，这些问题对高阶多项式的影响是很严重的，要特别注意。下面的例子介绍怎样简单地改动单项式基本插值公式来求得新的插值函数，新求出的插值函数能产生明显的改进效果。

例10.5 使用变换日期来对油价进行插值

因为元素数量级在$O(1)$和$O(10^{16})$之间变化，故上例中的Vandermonde矩阵是病态的。如果对应于插值x轴的日期重新调节，矩阵的条件数就可以明显改进。假设价格不是作为year（$1986\leq$ year ≤1996）的函数，而是作为变换日期（ys=year－1991）的函数进行插值，则 531

```
>> year = [1986   1988   1990   1992   1994   1996 ]';    % input data
>> price = [133.5  132.2  138.7  141.5  137.6  144.2]';
>> ys = year - mean(year);          % Shift dates by average value (1991)
>> A = vander(ys);                  % Set up Vandermonde system
>> c = A\price                      % and solve it
c =
    0.0030
    0.0374
   -0.0930
   -1.0237
    1.4899
  141.0863
```

变换日期的Vandermonde方程组不是病态的（如何与上例的Vandermonde方程组进行定量比较？）。另外，插值多项式的系数在5阶的数量级中变化，而不是像变换前那样在16阶数量级中变化。图10-8中是新产生的插值函数图，它没有图10-7中明显的乱真振荡。

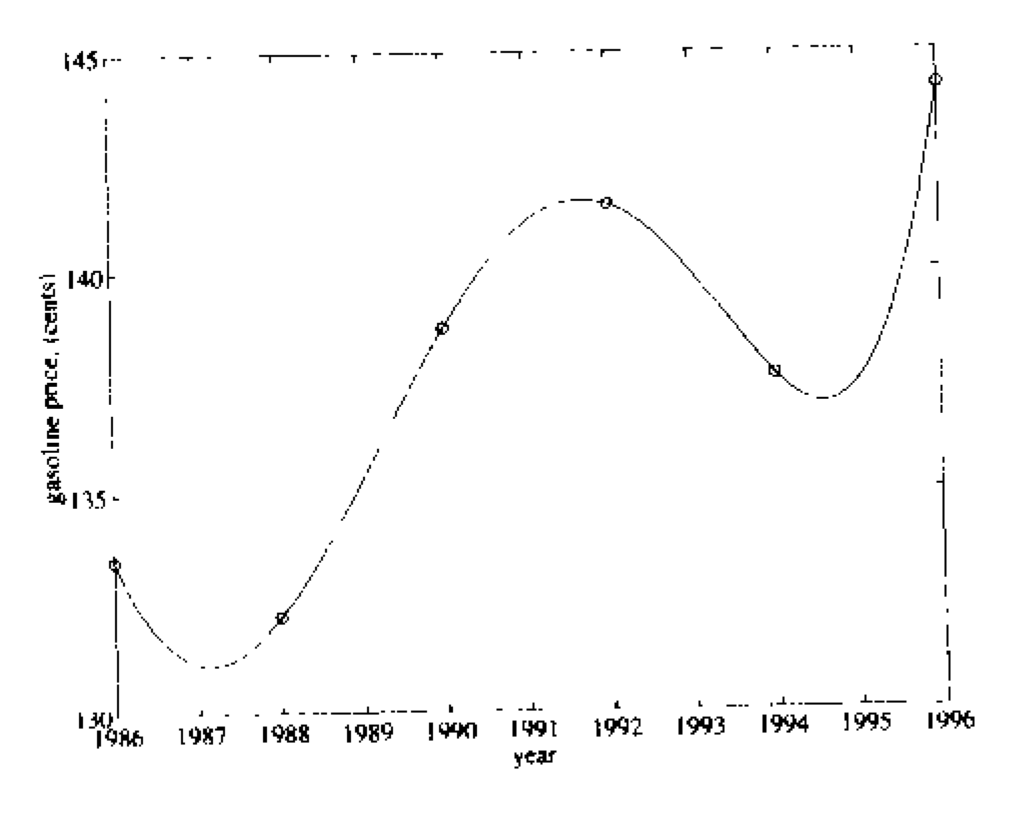


图10-8 汽油价格数据使用单项式基本插值公式和变换日期的插值

若不用变换日期而是使用规格化日期，单项式基本插值公式对汽油价格的插值将会得到进一步改进，这将在练习4中进行探讨。

10.2.2 用拉格朗日基本插值公式进行多项式插值

在单项式基本插值公式中通过点(x_1,y_1)和(x_2,y_2)的线性插值多项式为

532

$$P_1(x) = c_1x + c_2 \tag{10-9}$$

其中两个常数是

$$c_1 = \frac{y_2 - y_1}{x_2 - x_1} \qquad c_2 = \frac{y_1x_2 - y_2x_1}{x_2 - x_1}$$

将c_1和c_2代入方程（10-9)并进行整理，得

$$P_1(x) = y_1\frac{x - x_2}{x_1 - x_2} + y_2\frac{x - x_1}{x_2 - x_1}$$

这表明是用一对新的基本函数$L_1(x)$和$L_2(x)$来表示的线性插值多项式：

$$P_1(x)=y_1L_1(x)+y_2L_2(x) \tag{10-10}$$

其中

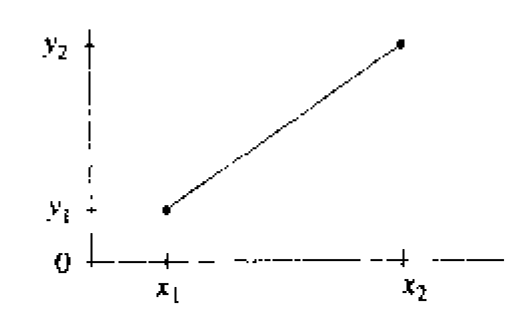
$$L_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad L_2(x) = \frac{x - x_1}{x_2 - x_1} \tag{10-11}$$

$L_1(x)$和$L_2(x)$是一阶拉格朗日插值多项式（Lagrange interpolating polynomial），图10-9中画出了$L_1(x)$和$L_2(x)$。这两个基本公式都是x的线性函数，而单项式基本插值公式包含线性函数x和常数“1”。如果x_i是一个支点，那么

$$L_i(x_j) = \delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{10-12}$$

其中δ_{ij}称为Kronecker delta函数。

533



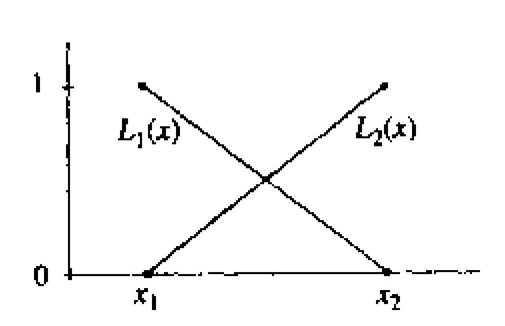


图10-9 一阶线性插值和相应的拉格朗日多项式插值

使用拉格朗日基本插值公式并穿过点(x_1,y_1)、(x_2,y_2)和(x_3,y_3)的二阶插值多项式为：

$$P_2(x) = y_1L_1(x) + y_2L_2(x) + y_3L_3(x)$$

其中

$$L_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}, \qquad L_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}$$

$$L_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

这里要预先警告读者的是，虽然没有下标或其他辨认标志，但是二阶插值多项式的L_1、L_2和L_3也可以与线性插值多项式的L_1和L_2区别开来。不给拉格朗日插值多项式指派阶数是一种惯例。

一般地，拉格朗日基本插值公式中的$n-1$阶多项式为

$$P_{n-1}(x)=y_1L_1(x)+y_2L_2(x)+\cdots+y_nL_n(x)=\sum_{j=1}^{n}y_jL_j(x) \tag{10-13}$$

其中

$$L_j(x)=\prod_{\substack{k=1\\k\neq j}}^{n}\frac{x-x_k}{x_j-x_k} \tag{10-14}$$

拉格朗日多项式与单项式基本函数插值多项式相比有两个重要的优点。首先，建立插值多项式不需要求解方程组。其次，拉格朗日多项式的估计值受舍入的影响要小得多。然而，遗憾的是方程（10-14）和方程（10-13）没有为$P_n(x)$提供很有效的数值解（与例10.10比较）。对已知支点集合来说，计算几个x值的拉格朗日插值函数需要对基本函数反复求值。拉格朗日多项式在多项式的符号运算中是非常有用的，其中方程（10-14）的简洁以及方程（10-12）中的特性是两个突出的特点。

插值多项式的惟一性 n阶多项式可用拉格朗日基本插值公式和单项式基本插值公式表示。关键的数学问题是这两个基本插值公式是否会导致不同的多项式，答案是否定的。[534]

> 穿过n个支点的$n-1$阶多项式是惟一的

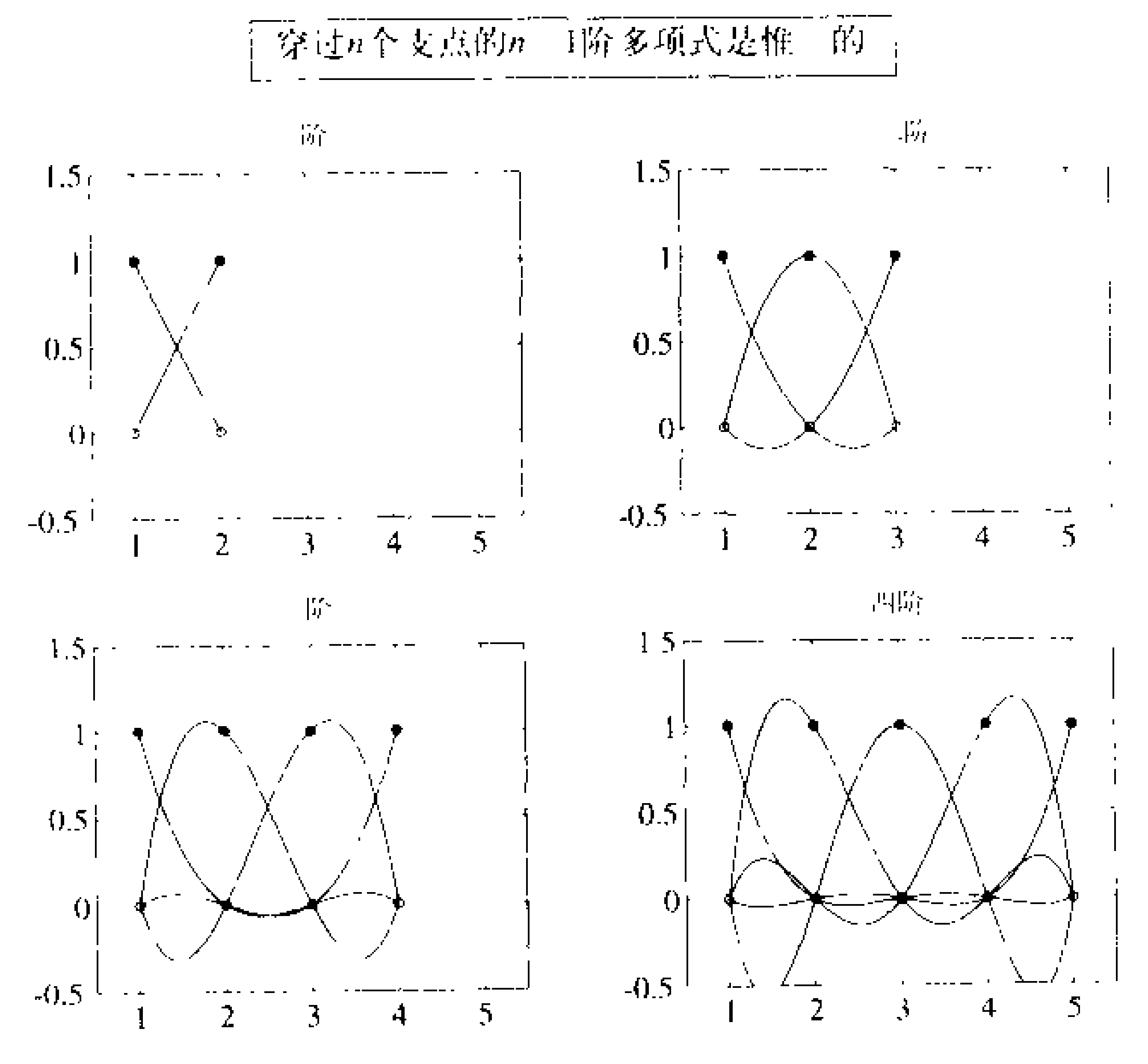


图10-10 一阶到四阶拉格朗日基本插值多项式

上述结论的证明，可参考Conte和de Boor [11,2.2节]或Stoer和Bulirsch[70.2.1节]。多项式的惟一性表示穿过(x_i,y_i)的$n-1$阶多项式必须相同（$i=1,\cdots,n$），不管它是用单项式基本插值公

式、拉格朗日基本插值公式还是用其他多项式基本插值公式表示。每个基本函数的系数当然会不同，但两个多项式表达式所产生的值在严格的算术意义上是一致的。但是，如第5章所述，在严格的算术意义上一致的表达式在浮点运算中不一定会产生相同的结果。这个结论从例10.4到例10.6中可以很明显地看出来。

实现 为计算已知阶数的拉格朗日插值，需要用方程（10-14）计算拉格朗日基本函数，再用方程（10-13）求拉格朗日多项式的值。MATLAB中有一种高效率的实现，需要对一些数

535

（量）进行向量化和预先计算。

方程（10-14）可记为

$$L_j(x)=\prod_{\substack{k=1\\k\neq j}}^{n}\frac{x-x_k}{x_j-x_k}=\left[\prod_{k=1}^{j-1}\frac{x-x_k}{x_j-x_k}\right]\left[\prod_{k=j+1}^{n}\frac{x-x_k}{x_j-x_k}\right] \tag{10-15}$$

给定x的值，每个L_j（分数）的分子包含了下列各项的乘积：

$$(x-x_1)(x-x_2)\ldots(x-x_{j-1})\,(x-x_{j+1})\ldots(x-x_n)$$

因为这些项在每个L_j分数中重复，所以可只计算一次差值并将结果存储起来。令dxi为包含所有可能的$(x-x_k)$值的向量：

```
xi  = ...        % Evaluate interpolant at xi (given)
x   = ...        % Tabulated x data (given)
dxi = xi - x;    % Vector of xi - x(k) values
```

已知dxi，我们可以用下列语句求方程（10-15)中的分子：

```
num = prod(dxi(1:j-1))*prod(dxi(j+1:n))
```

其中，内置prod函数用来计算向量元素的乘积，子表达式prod(dxi(1:j-1))等价于$(x-x_1)(x-x_2)\ldots(x-x_{j-1})$。

方程（10-15）中的分母也可以从预先计算量中得到，但这样做并不好，因为它需要存储下三角矩阵，此矩阵中的元素仅使用两次（见练习9）。如果分母只在需要时计算，那么计算工作量就会比最小工作量的两倍还要多，但不需额外的存储空间。在实际中，因为只有中等阶数的多项式用得比较多，所以这里并不苛求在（较少）内存和（较多）计算量间做平衡。采用简单易懂但相对低效的方法，我们可以使用如下语句求出方程（10-15）中的分母：

```
den = prod(x(j)-x(1:j-1))*prod(x(j)-x(j+1:n));
```

其中，x(j)-x(1:j-1)是差值x(j)-x(1),x(j)-x(2),...,x(j)-x(j-1)的向量。

程序清单10-1中的函数lagrint用前面的公式计算穿过一个用户指定数据集的拉格朗日

536

多项式。注意lagrint函数一次只能对一个标量xi插值。

程序清单10-1 用拉格朗日多项式进行插值的lagrint函数

```
function yi = lagrint(x,y,xi)
% lagrint  Interpolation with Lagrange polynomials of arbitrary degree
%
% Synopsis:  yi = lagrint(x,y,xi)
%
% Input:   x,y = tabulated data
%          xi  = point where interpolation is to be evaluated
%
% Output:    yi = value of y at x = xi obtained via interpolation with
%                 polynomial of degree n-1, where length(y) = length(x) = n
%
```

```
dxi = xi - x;          % vector of xi - x(1), xi - x(2), ... values
n = length(x);         % degree of polynomial is n-1
L = zeros(size(y));    % preallocate L for speed

% Refer to section 10.2.2 in text for explanation of vectorized code
% used to compute Lagrange basis functions, L(j)
L(1) = prod(dxi(2:n))/prod(x(1)-x(2:n));        % j = 1
L(n) = prod(dxi(1:n-1))/prod(x(n)-x(1:n-1));    % j = n
for j=2:n-1
  num = prod(dxi(1:j-1))*prod(dxi(j+1:n));
  den = prod(x(j)-x(1:j-1))*prod(x(j)-x(j+1:n));
  L(j) = num/den;
end
yi = sum(y.*L);        % Evaluate Polynomial: sum of y(j)*L(j), j=1..n
```

例10.6 用拉格朗日多项式对汽油价格进行插值

程序清单10-2中的demoGasLag函数使用拉格朗日多项式对例10.4中的汽油价格数据进行插值。所得到的插值函数图形（这里没有给出）与图10-8看起来非常类似（看不出有区别）。特别地，尽管在demoGasLag中没有使用变换日期，插值函数中也不会出现乱真振荡，这是因为拉格朗日多项式不需要线性方程组的解，也就不会有破坏Vandermonde方程组解的精度的病态性和舍入误差问题。读者可以回想一下，在严格的算术意义上，拉格朗日插值与单项式基本插值得到的多项式是一样的，但是，正如本例所示，用拉格朗日基本插值公式比用单项式基本插值公式进行多项式插值受舍入误差影响更小。

537

程序清单10-2 使用lagrint函数对汽油价格数据进行插值

```
function demoGasLag
% demoGasLag  Interpolate gasoline price data with Lagrange polynomials
%
% Synopsis:  demoGasLag
%
% Input:     none
%
% Output:    Plot of given data and interpolating function for gas price

year =  [1986   1988   1990   1992   1994   1996 ]';    % input data
price = [133.5  132.2  138.7  141.5  137.6  144.2]';

y = linspace(min(year),max(year),200);  % eval interpolant at these dates
p = zeros(size(y));                     % Pre-allocate p for efficiency
for i=1:length(y)
  p(i) = lagrint(year,price,y(i));      % Interpolate to find p( y(i) )
end
plot(year,price,'o',y,p,'-');
xlabel('year');  ylabel('gasoline price, (cents)');
```

10.2.3 使用牛顿基本插值公式进行多项式插值

n阶插值多项式的牛顿形式为:

$$P_n(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + \cdots + c_{n+1}(x - x_1)(x - x_2)\cdots(x - x_n)$$

其中，牛顿基本多项式为

$$1,\ (x - x_1),\ (x - x_1)(x - x_2),\ (x - x_1)(x - x_2)(x - x_3),\ \ldots$$

c_i是在条件$P_n(x_i) = f(x_i)$，$i = 1,\ldots,n+1$下求得的系数。虽然初看起来这些基本函数可能很繁琐，但最终的牛顿插值公式在计算上要比单项式基本插值公式和拉格朗日基本插值公式更加有效。牛顿插值公式也有很好的数值特性，它对第11章讨论的插值方案的理论分析和数值积分法也非常有用。

使用牛顿插值多项式运算时，使用均差法（divided-difference notation、差分法）进行计算会更容易。我们先不介绍最一般形式的均差法，而是从简单的二阶多项式情况入手，然后将二阶插值多项式推广到三阶插值多项式。容易扩展插值多项式的阶数是牛顿插值多项式的一个优点。介绍完这些例子后，将会列出均差法的一般特性，并给出牛顿插值多项式的MATLAB实现。

[538]

二阶多项式的牛顿形式　考虑通过(x_i, y_i)的二阶多项式（$i = 1,2,3$），它以牛顿基本函数表示为：

$$P_2(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) \tag{10-16}$$

应用三个约束条件，得

$$P_2(x_1) = c_1 = y_1$$
$$P_2(x_2) = c_1 + c_2(x_2 - x_1) = y_2$$
$$P_2(x_3) = c_1 + c_2(x_3 - x_1) + c_3(x_3 - x_1)(x_3 - x_2) = y_3$$

或者用矩阵形式表示为：

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & (x_2 - x_1) & 0 \\ 1 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \tag{10-17}$$

解此方程组可得出方程（10-16）中多项式的系数。因为方程组是下三角矩阵，它的求解需要的运算量就是$O(n^2)$，而不是满系数矩阵的Vandermonde方程组的$O(n^3)$。

方程（10-16）中的c_i可以由简洁的公式给出，此公式就称为均差法。为促成这些表达式，需要在矩阵形式下求解前面的3 × 3方程组（另见[68、77]）。这样就可以自然地导出均差公式。

检查方程（10-17）可知$c_1 = y_1$。用向前代入法可产生计算c_2和c_3的公式。首先，从第二、第三行中减去第一行，得

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & (x_2 - x_1) & 0 \\ 0 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 - y_1 \\ y_3 - y_1 \end{bmatrix}$$

[539]

然后，将第二行除以$x_2 - x_1$，第三行除以$x_3 - x_1$，得：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 1 & (x_3-x_2)\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}y_1\\\dfrac{y_2-y_1}{x_2-x_1}\\\dfrac{y_3-y_1}{x_3-x_1}\end{bmatrix} \tag{10-18}$$

由第二行可得$c_2=(y_2-y_1)/(x_2-x_1)$。现在引入一阶均差：

$$f[x_1,x_2]\equiv\frac{y_2-y_1}{x_2-x_1}\quad 和\quad f[x_1,x_3]\equiv\frac{y_3-y_1}{x_3-x_1}$$

一般地，有序对(x_i, y_i)和(x_j, y_j)的一阶均差为：

$$f[x_i,x_j]\equiv\frac{y_j-y_i}{x_j-x_i} \tag{10-19}$$

引入这种新的表示法后，$c_2=f[x_1, x_2]$，方程（10-18）就变成了：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 1 & (x_3-x_2)\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}y_1\\f[x_1,x_2]\\f[x_1,x_3]\end{bmatrix}$$

第三行减去第二行得：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & (x_3-x_2)\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}y_1\\f[x_1,x_2]\\f[x_1,x_3]-f[x_1,x_2]\end{bmatrix}$$

第三行除以x_3-x_2得：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}\begin{bmatrix}c_1\\c_2\\c_3\end{bmatrix}=\begin{bmatrix}y_1\\f[x_1,x_2]\\f[x_1,x_2,x_3]\end{bmatrix}$$

其中，经过代数变换得：

$$\frac{f[x_1,x_3]-f[x_1,x_2]}{x_3-x_2}=\frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_2}\equiv f[x_1,x_2,x_3]$$

$f[x_1, x_2, x_3]$是涉及(x_1,y_1)、(x_2,y_2)和(x_3, y_3)的二阶均差。于是，方程（10-17）的解为：

$$\begin{aligned}c_1&=y_1\\c_2&=f[x_1,x_2]\\c_3&=f[x_1,x_2,x_3]\end{aligned} \tag{10-20}$$

540

习惯上，零阶均差定义为

$$f[x_i]=y_i$$

所以，$y_1=f[x_1]$，二阶多项式的牛顿插值形式可记为：

$$P_2(x)=f[x_1]+f[x_1,x_2](x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2) \tag{10-21}$$

从方程可以看出，牛顿多项式的系数就是均差。

三阶多项式的牛顿插值形式 三阶多项式的牛顿插值可由二阶多项式的直接扩展获得。其三阶多项式为:

$$P_3(x) = c_1 + c_2(x - x_1) + c_3(x - x_1)(x - x_2) + c_4(x - x_1)(x - x_2)(x - x_3) \tag{10-22}$$

通过重复分析前面部分求解出c_i（参照练习10），得

$$\begin{aligned} c_1 &= y_1 \\ c_2 &= f[x_1, x_2] \\ c_3 &= f[x_1, x_2, x_3] \\ c_4 &= f[x_1, x_2, x_3, x_4] \end{aligned} \tag{10-23}$$

其中

$$f[x_1, x_2, x_3, x_4] = \frac{f[x_2, x_3, x_4] - f[x_1, x_2, x_3]}{x_4 - x_1}$$

且

$$f[x_2, x_3, x_4] = \frac{f[x_3, x_4] - f[x_2, x_3]}{x_4 - x_2}$$

注意c_1、c_2和c_3就是方程（10-20）中的c_1、c_2和c_3。将均差系数代入方程（10-22）得:

$$\begin{aligned} P_3(x) = {} & f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) \\ & + f[x_1, x_2, x_3, x_4](x - x_1)(x - x_2)(x - x_3) \end{aligned} \tag{10-24}$$

比较方程（10-21）和方程（10-24）有

541

$$P_3(x) = P_2(x) + f[x_1, x_2, x_3, x_4](x - x_1)(x - x_2)(x - x_3)$$

此特性适用于任何n阶和$n-1$阶多项式对，即

$$P_n(x) = P_{n-1}(x) + f[x_1, x_2, \ldots, x_{n+1}](x - x_1)(x - x_2)\cdots(x - x_n) \tag{10-25}$$

由方程（10-25）可以看出，牛顿插值多项式可以逐渐地更新，以确定增加多项式的阶数是否会提高插值精度。因为低阶数多项式的系数可以不加更改地使用，所以这种方法在计算上很有效。

均差特性 牛顿多项式使用均差表示非常方便。作为参考，这里列出了均差的一些重要特性，可参考[10、11、40、68、70]中的例子作为验证。注意，本书中n阶多项式是定义在支点(x_i, y_i)（$i = 1,\ldots,n+1$）集合上的。在关于数值计算的绝大多数书中，$n+1$个支点的下标范围是$i = 0,\ldots,n$。下标从$i = 0$开始使标记变得很方便，但MATLAB中的向量和矩阵下标从1开始，二者有冲突。下面的公式就与MATLAB中的惯用下标一致。

1. 均差$f[x_1, x_2, \ldots, x_k]$是点(x_1, f_1)、(x_2, f_2)、…，(x_k, f_k)的插值多项式中项x^{k-1}的系数，因此n阶插值多项式的牛顿插值公式可记为:

$$\begin{aligned} P_n(x) = {} & f[x_1] + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2) \\ & + f[x_1, x_2, \ldots, x_{n+1}](x - x_1)(x - x_2)\cdots(x - x_n) \end{aligned} \tag{10-26}$$

2. n阶插值多项式可通过向$n-1$阶牛顿多项式添加一个更高次项获得，即:

$$P_n(x) = P_{n-1}(x) + f[x_1, x_2, \ldots, x_{n+1}](x - x_1)(x - x_2)\cdots(x - x_n)$$

其紧凑表示为，

$$P_n(x) = P_{n-1}(x) + f[x_1, x_2, \cdots, x_{n+1}]\prod_{i=1}^{n}(x - x_i) \qquad (10\text{-}27)$$

3. 若$f(x)$为k阶多项式，则$k+2$、$k+3$、...阶均差都等于零。

4. 均差序列可由下列公式递归得到：

$$f[x_1, \cdots, x_k] = \frac{f[x_2, \cdots, x_k] - f[x_1, \cdots, x_{k-1}]}{x_k - x_1} \qquad (10\text{-}28)$$

另外，零阶均差定义为

$$f[x_i] = y_i \qquad (10\text{-}29)$$

[542]

5. 均差$f[x_1, \ldots, x_k]$不会因为自变量参数$x_1, \cdots, x_k$排列的不同而改变。

例10.7 使用牛顿基本公式进行多项式手工插值

采用二次插值法和例10.2中的数据可以估算甘油在22℃时的粘度。插值形式为：

$$\mu(T) = f[T_1] + f[T_1, T_2](T - T_1) + f[T_1, T_2, T_3](T - T_1)(T - T_2)$$

在作任何计算之前，必须决定用哪三个支点来定义插值式。在T = 22℃下，"最近"的三个点是$T_1 = 10$, $T_2 = 20$和$T_3 = 30$，相关的均差为：

$$f[T_1] = \mu_1 = 3.810$$

$$f[T_1, T_2] = \frac{\mu_2 - \mu_1}{T_2 - T_1} = \frac{1.492 - 3.810}{10} = -0.2318$$

$$f[T_2, T_3] = \frac{\mu_3 - \mu_2}{T_3 - T_2} = \frac{0.629 - 1.492}{10} = -0.0863$$

$$f[T_1, T_2, T_3] = \frac{f[T_2, T_3] - f[T_1, T_2]}{T_3 - T_1} = \frac{-0.0863 + 0.2318}{20} = 7.2750 \times 10^{-3}$$

于是插值公式变为：

$$\mu(T) = 3.810 - 0.2318(T - 10) + 7.2750 \times 10^{-3}(T - 10)(T - 20)$$

在T = 22℃下对插值式求值，得

$$\mu(22) = 3.810 - 0.2318(12) + 7.2750 \times 10^{-3}(12)(2) = 1.2030$$

结果取决于支点的选取和插值多项式的阶数（参见例10.2和练习12、13）。

均差表 方程（10-28）提供了求解任意均差系数的简洁公式。方程是递归的：高阶系数由两个低一阶的系数的差值（差分）决定，低一级系数又由两个更低一阶的系数的差值决定，递归到只剩零阶差（参考方程（10-29））时结束。例如，求解$f[x_1, x_2, x_3]$值的过程可表示为：

$$
\begin{array}{ccccc}
 & & f[x_1,x_2,x_3] & & \\
 & \swarrow & & \searrow & \\
f[x_1,x_2] & & & & f[x_2,x_3] \\
\swarrow \quad \searrow & & & & \swarrow \quad \searrow \\
f[x_1] \qquad & & f[x_2] & & \qquad f[x_3]
\end{array}
$$

[543]

零阶均差是函数在支点的值（例如$f[x_1] = y_1$）。均差系数的递归定义很简洁，但它不提供求解这些系数的计算过程。

均差系数可以在均差表的辅助下手工计算。均差表将计算组织成高阶系数依赖低阶系数的形式。在$f[x_1, x_2, x_3]$的示意图中，第一列的零阶均差已知。为计算$f[x_1, x_2, x_3]$，需要上溯依赖（相关）树，相同的信息组织成表10-1中的列形式。前两列是已知的(x_i, y_i)数据。第三列包含了一阶均差，它决定于前面列的数据。高阶差分由其左边的低阶差分求得。表10-1中的箭头表示计算$f[x_1, x_2, x_3]$的均差的顺序。

表10-1　均差表

x_i	$f[\,]$	$f[,]$	$f[,,]$	$f[,,,]$
x_1	$f[x_1]$			
x_2	$f[x_2]$	$f[x_1, x_2]$		
x_3	$f[x_3]$	$f[x_2, x_3]$	$f[x_1, x_2, x_3]$	
x_4	$f[x_4]$	$f[x_3, x_4]$	$f[x_2, x_3, x_4]$	$f[x_1, x_2, x_3, x_4]$

例10.8　FET电流特性的手工插值

本例给出了构造均差表所涉及的详细计算。下表给出了一个场效应管(FET)的漏极电流I_d和漏源极间电压V_{ds}的关系数据，前者是后者的函数。

V_{ds}(V)	0	0.4	0.75	1.3	2	3	4.5	5
I_d(mA)	0	4.95	10.14	15.0	17.6	19.05	20.32	20.5

用三阶插值法来求V_{ds}=1.5V处的I_d。

电流I是电压V的函数，其三阶牛顿多项式为：

$$I(V) = f[V_1] + f[V_1, V_2](V - V_1) + f[V_1, V_2, V_3](V - V_1)(V - V_2)$$
$$+ f[V_1, V_2, V_3, V_4](V - V_1)(V - V_2)(V - V_3)$$

离V=1.5最近的四个支点是V = 0.4、0.75、1.3和2。将已知支点数据输入前两列，开始构544造均差表：

V_i	$f[V_i]$	$f[,]$	$f[,,]$	$f[,,,]$
0.40	4.95			
0.75	10.14	$f[V_1, V_2]$		
1.30	15.00	$f[V_2, V_3]$	$f[V_1, V_2, V_3]$	
2.00	17.60	$f[V_3, V_4]$	$f[V_2, V_3, V_4]$	$f[V_1, V_2, V_3, V_4]$

由公式

$$f[V_{i-1}, V_i] = \frac{I_i - I_{i-1}}{V_i - V_{i-1}}$$

填入第三列的值，例如，

$$f[V_1, V_2] = \frac{10.14 - 4.95}{0.75 - 0.4} = 14.8286$$

完成第三列数据，得

V_i	$f[V_i]$	$f[,]$	$f[,,]$	$f[,,,]$
0.40	4.95			
0.75	10.14	14.8286		
1.30	15.00	8.8364	$f[V_1,V_2,V_3]$	
2.00	17.60	3.7143	$f[V_2,V_3,V_4]$	$f[V_1,V_2,V_3,V_4]$

第四和第五列填入值的计算公式分别为:

$$f[V_{i-2},V_{i-1},V_i]=\frac{f[V_{i-1},V_i]-f[V_{i-2},V_{i-1}]}{V_i-V_{i-2}}$$

以及

$$f[V_{i-3},V_{i-2},V_{i-1},V_i]=\frac{f[V_{i-2},V_{i-1},V_i]-f[V_{i-3},V_{i-2},V_{i-1}]}{V_i-V_{i-3}}$$

例如

$$f[V_1,V_2,V_3]=\frac{8.8364-14.8286}{1.3-0.4}=-6.6580$$

完整的均差表是:

V_i	$f[V_i]$	$f[,]$	$f[,,]$	$f[,,,]$
0.40	4.95			
0.75	10.14	14.8286		
1.30	15.00	8.8364	6.6580	
2.00	17.60	3.7143	4.0977	1.6002

545

插值多项式的系数在均差表的对角线上(参见表10.1)。故:

$$I(V)=4.95+14.8286(V-V_1)-6.6580(V-V_1)(V-V_2)$$
$$+1.6002(V-V_1)(V-V_2)(V-V_3)$$

计算$V=1.5$V处的多项式,得

$$I(V)=4.95+14.8286(1.1)-6.6580(1.1)(0.35)+1.6002(1.1)(0.35)(0.2)$$
$$=16.0326\,\text{mA}$$

均差表的自动计算 虽然前面例子中的手工计算很直观,但计算过程是冗长乏味的。程序清单10-3中的divDiffTable函数可以自动生成均差表,此表存储在矩阵中,且其中的元素按列计算。矩阵的第一列包含$y=f(x)$的值,这些值是零阶均差。此列由下列语句赋值:

```
D(:,1) = y(:);
```

程序清单10-3 函数divDiffTable产生均差表

```
function D = divDiffTable(x,y)
% divdiffTable  Construct a table of divided-difference coefficients
%
% Synopsis:   D = divDiffTable(x,y)
%
% Input:      x,y = vectors containing the tabulated y = f(x) data
%
% Output:     D = matrix containing divided-difference coefficients in
```

```
%                its lower triangle.  Diagonal entries are coefficients
%                of the Newton polynomial
n = length(y);
if length(x)~=n,   error('x and y are not compatible');   end

D = zeros(n,n);
D(:,1) = y(:);    %  First column is zeroth order difference, f[x_i] = y_i
for j=2:n
  for i=j:n
    D(i,j) = (D(i,j-1)-D(i-1,j-1))/(x(i)-x(i-j+1));
  end
end
```

546

表达式y(:)保证了等号右边的结果为一个列向量。第二列到第n列的对角线及其下面的元素使用下列公式计算:

```
D(i,j) = (D(i,j-1)-D(i-1,j-1))/(x(i)-x(i-j+1));
```

注意，上面表达式的分母不是x(i)-x(i-1)。

例10.8的结果由divDiffTable函数可以很简单地求出来，过程如下:

```
>> Vds = [0.4    0.75   1.3    2  ];    %  Define Id = f(Vds) data
>> Id  = [4.95   10.14  15.0   17.6];
>> D = divDiffTable(Vds,Id)            %  Construct divided difference table
D =
    4.9500          0          0          0
   10.1400    14.8286          0          0
   15.0000     8.8364    -6.6580          0
   17.6000     3.7143    -4.0977     1.6002

>> c = diag(D);        %  Extract coeffients of Newton polynomial
c =
    4.9500
   14.8286
   -6.6580
    1.6002

>> v = 1.5;
>> Iv = c(1) + c(2)*(v-Vds(1)) + c(3)*(v-Vds(1))*(v-Vds(2)) ...
      + c(4)*(v-Vds(1))*(v-Vds(2))*(v-Vds(3))
Iv =
   16.0326
```

newtint函数　对牛顿多项式的常规插值来说，只有均差表中的对角线上的元素才有用，这里用牛顿多项式构造了一个称为newtint的m文件函数，对给定数据集进行插值。函数newtint有两个任务：构造均差表的对角线上的元素，以及计算牛顿多项式的值。

因为高阶均差由低阶均差决定，所以就不可能只计算均差表对角线上的元素，对角线上的元素必须从低阶均差求出，但不需要存储所有的中间结果。

547

考虑用牛顿基本公式创建的三阶多项式。为简便起见，将多项式记为:

$$P_3(\hat{x}) = c_1 + c_2(\hat{x}-x_1) + c_3(\hat{x}-x_1)(\hat{x}-x_2) + c_4(\hat{x}-x_1)(\hat{x}-x_2)(\hat{x}-x_3) \tag{10-30}$$

其中系数c_i由对应的均差给出（例如$c_3 = f[x_1,x_2,x_3]$），$\hat{x}$是要计算插值多项式的x值。现在的目标是设计一个算法，不需要构造整个矩阵就可以计算出c_i的值，并且能析取出对角线上的

元素。为达到这个目标，可以将中间（低阶）均差存储在向量c中，并根据需要覆盖c中的适当元素，这个过程如下表所示：

x_i	$f[\,]$	$f[,]$	$f[,,]$	$f[,,,]$
x_1	c_1			
x_2	c_2	c_2		
x_3	c_3	c_3	c_3	
x_4	c_4	c_4	c_4	c_4

表中对角线上的粗体字是c_i的最终值，对角线下的c_i值在算法执行中被逐渐覆盖。

在计算的开始阶段，c_i被赋给表中第二列的零阶均差。在计算的结尾，c_i包含均差表中的对角线元素。每个连续列都在for循环中处理，其中c的一个元素被代替为：

$$c_i \leftarrow \frac{c_i - c_{i-1}}{x_i - x_{i-j+1}}$$

此计算与divDiffTable表中的$D(i,j)$的计算一致。既然c_i将被覆盖，那么就需要对对角线及其以下的表中元素进行一次扫描。否则，c_{i-1}的值就会在它用于计算c_i之前被改变。前面的思想包含在下面代码的循环中：

```
n = length(y);
c = y(:);          % First column is zeroth-order difference, f[x_i] = y_i
for j=2:n
  for i=n:-1:j   % Work backward to keep from overwriting unused data
    c(i) = (c(i)-c(i-1))/(x(i)-x(i-j+1));
  end
end
```

548

一旦c_i已知，必须对牛顿多项式求值。注意，不能使用内置polyval函数(为什么不能？)。简便起见，考虑方程（10-30）中的三阶多项式，该多项式的计算可以组织成嵌套相乘形式[⊖](nested multiplication form)，如下：

$$P_3(\hat{x}) = c_1 + (\hat{x}-x_1)(c_2 + (\hat{x}-x_2)(c_3 + c_4(\hat{x}-x_3))) \qquad (10\text{-}31)$$

方程可以用单循环表示，此循环从最里面的乘积$c_4(\hat{x}-x_3)$开始，对任意长度的c，MATLAB中的实现为：

```
yhat = c(n);
for i=n-1:-1:1
  yhat = yhat.*(xhat-x(i)) + c(i);
end
```

程序清单10-4中的函数newtint包含了以上代码段。函数newtint可以使用以下三种方法来调用：

```
yhat      = newtint(x,y,xhat)
[yhat,dy] = newtint(x,y,xhat)
[yhat,dy,c] = newtint(x,y,xhat)
```

输入参数x和y是定义了要插值的表格数据向量，输入参数xhat是在该处要计算插值的标

⊖ 这类似于Horner公式中使用单项式基本插值公式来计算多项式（参见例3.7）。

量或向量值。插值的结果返回给yhat。如果xhat是一个向量，那么yhat也是一个向量。之所以能够这么灵活，是因为向yhat的嵌套计算中添加了一个矩阵操作符。注意，插值多项式的阶数由x和y的向量长度隐式指定。如果n=length(y)，那么yhat就在n-1阶多项式中计算。

可选输出参数是dy和c。dy是yhat中$n-1$和$n-2$阶多项式插值的差。c是牛顿多项式的系数。既然牛顿多项式是递归建立的，那么就很容易由计算yhat的c计算出dy的值。

例10.9　用牛顿多项式对汽油价格进行插值

程序清单10-5中的demoGasNewt函数重复了例10.4和例10.6中的汽油价格插值。运行demoGasNewt可得到图10-8。

549

程序清单10-4　函数newtint使用均差进行多项式插值

```
function [yhat,dy,cout] = newtint(x,y,xhat)
% newtint  Interpolation with Newton polynomials of arbitrary degree
%
% Synopsis:  yhat        = newtint(x,y,xhat)
%            [yhat,dy]   = newtint(x,y,xhat)
%            [yhat,dy,c] = newtint(x,y,xhat)
%
% Input:     x,y = vectors containing the tabulated y = f(x) data
%            xhat = (scalar or vector) x values where interpolant is evaluated
%
% Output:    yhat = value of y at each xhat obtained via interpolation
%            dy   = (optional) change in value of interpolant between
%                   polynomials of degree m-1 and m.  m = n-1 = length(x)-1
%            c    = (optional) coefficients of the Newton form of polynomial
%
% Note:      Degree of interpolating polynomial is implicitly specified
%            by the length of the x and y vectors.  If n = length(y) then
%            yhat is evaluated with a polynomial of degree (n-1)

n = length(y);   if length(x)~=n,  error('x and y are not compatible');  end

% --- Construct polynomial coefficients from diagonal of div.-diff. table
c = y(:);         %  First column is zeroth-order difference, f[x_i] = y_i
for j=2:n
  for i=n:-1:j   %  Work backward to keep from overwriting unused data
    c(i) = (c(i)-c(i-1))/(x(i)-x(i-j+1));
  end
end

% --- Nested evaluation of the polynomial
yhat = c(n);
for i=n-1:-1:1
  yhat = yhat.*(xhat-x(i)) + c(i);    %  Array op allows vector of xhats
end

% --- optional output
if nargout>1
  yn2 = c(n-1);      %  begin evaluation of polynomial of degree n-2
  for i=n-2:-1:1
```

```
    yn2 = yn2.*(xhat-x(i)) + c(i);
  end
  dy = yhat - yn2;  %  difference of interpolants of degree n-2 and degree n-1
  if nargout>2, cout = c;  end   %  copy coefficients to output variable
end
```

550

程序清单10-5 函数demoGasNewt用Newton多项式对假设的汽油数据进行插值。插值由NMM工具箱中的newtint函数完成

```
function demoGasNewt
% demoGasNewt  Interpolate gasoline price data with Newton polynomials
%
% Synopsis:  demoGasNewt
%
% Input      none
%
% Output:    Plot of given data and interpolating function for gas price

year = [1986   1988   1990   1992   1994   1996 ]';    %  input data
price = [133.5  132.2  138.7  141.5  137.6  144.2]';

% --- Interpolate and plot results
y = linspace(min(year),max(year),200);  %  eval interpolant at these dates
p = newtint(year,price,y);
plot(year,price,'o',y,p,'-');
xlabel('year');  ylabel('gasoline price, (cents)');
```

例10.10 比较各种基本公式构成的插值公式的计算量

在严格的算术意义上，给定数据集的插值与构造插值式的基本公式无关。但是，如例10.4和例10.5中所述，不同的多项式基本函数可能会有不同的数值特性。本例对由不同基本公式构成的插值公式的计算量进行了比较。

函数compInterp（此处未列出，包含在NMM工具箱中）计算出使用变形单项式、拉格朗日和牛顿基本公式时对例10.4中的汽油价格数据进行插值的浮点操作次数。已知有六个油价数据的有序对，我们可以计算出N个点上的五阶插值多项式。找几个N值，运行compInterp可得到如下结果：

基本多项式	不同N值的浮点操作次数				
	1	10	25	100	250
变形单项式	520	625	820	1795	3745
拉格朗日	141	1410	3525	14100	35250
牛顿	106	241	466	1591	3841

此数据的图形描述如图10-11所示。

当$N>100$，拉格朗日多项式基本插值公式要比其他基本插值公式多做一个数量级的工作。这是因为拉格朗日多项式的系数必须对每个数据点重新求值；对中小N值，获得相同的结果，牛顿基本插值公式所用的步骤明显较少；对小N值，采用单项式基本插值公式法进行插值的工作基本上花在解Vandermonde方程组上；对大N值，采用单项式基本插值公式或牛顿基本插值公式进行插值的工作基本上是求多项式的值。所以，这两种方法的浮点操作次数基本相同。

551

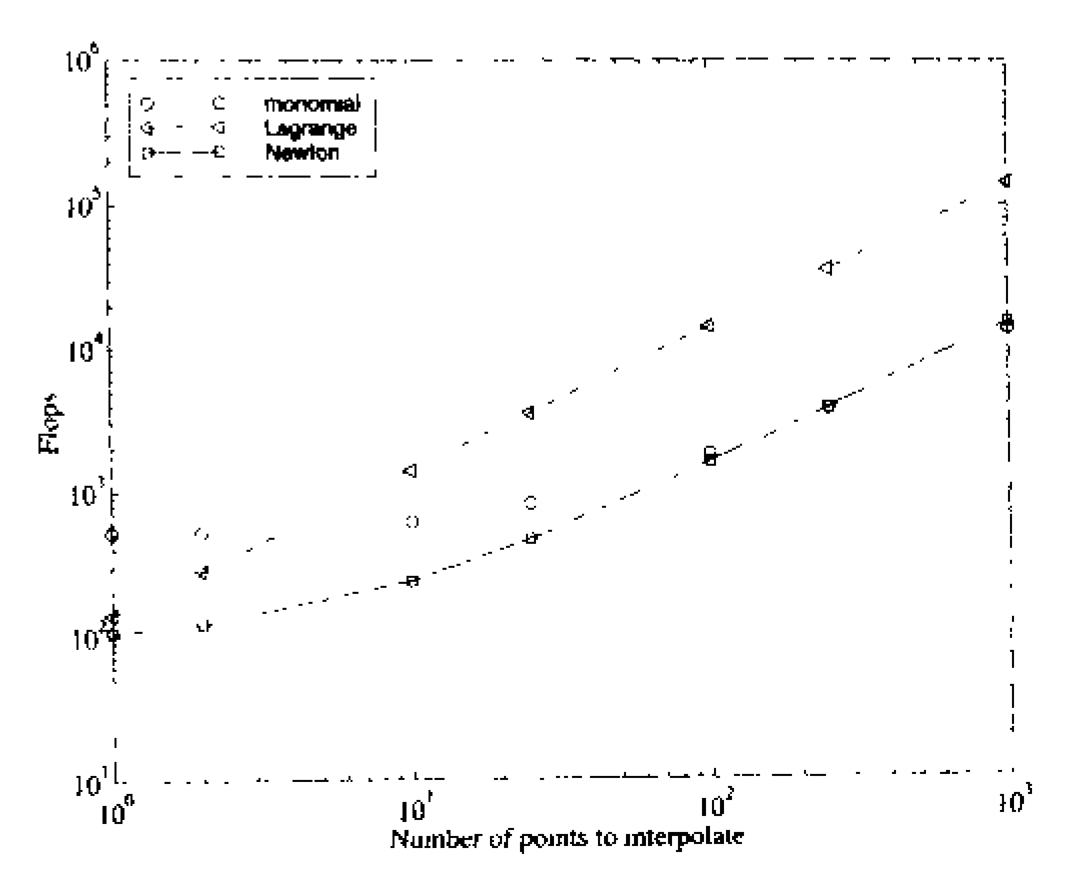


图10-11　使用不同多项式基本插值公式的浮点操作次数比较：对六个点的数据表进行五阶多项式插值

这些结果说明插值多项式的牛顿形式更好一些。它计算高效，并且随着多项式阶数的增加它仍能够保持比较可行的操作次数。当N较大时，求解单项式基本插值公式有微小的计算量上的优势，但就是这种优势也轻易地被Vandermonde方程组的病态性掩盖了，因为必须求解此方程组，才能求出单项式基本插值公式中多项式的系数。

10.2.4　多项式摆动

增加插值多项式的阶数并不一定能增加插值的精度。根据定义，插值式$F(x)$可以与支点(x_i,y_i)，$i=1,\ldots,n$处的实际函数匹配，但是却不能保证在支点之间，$F(x)$还能很好地逼近产生(x_i,y_i)数据的实际函数$f(x)$。例如，如果$f(x)$为一个已知的解析函数，且定义$F(x)$的支点集合中数据点的数目n可以增加（多项式$F(x)$的阶数也因此增加），那么插值式就可能在支点间振荡。甚至当实际函数$f(x)$平滑时，这种多项式摆动（*polynomial wiggle*）也可能发生。

552

图10-7中的明显高频振荡却不是由于多项式摆动引起的，而是由多项式的项相加来求插值多项式时发生的舍入误差造成的。

有时，多项式摆动可通过谨慎选择基础函数的取样点来减小（参见练习21和22）。但是如果数据是由不容易重复的实验中取得的，就不能这么做了，这时会用下一节介绍的分段插值法。在讲分段插值之前，先考虑下例中所演示的多项式摆动问题。

例10.11　有关多项式插值插动的演示

函数$y=f(x)$由两条直线$y=-0.5x+4$（$1\leqslant x\leqslant 5$）和$y=-0.4x$（$5<x\leqslant 10$）连接产生。此函数的十个样本值显示在下表中。在两个线性函数交接的地方函数$f(x)$有一个不大的台阶。

x_i	1	2	3	4	5	6	7	8	9	10
y_i	3.5	3.0	2.5	2.0	1.5	−2.4	−2.8	−3.2	−3.6	−4.0

由表中的样本数据，可以创建一个严格穿过所有十个点的九阶多项式。但由于多项式的

摆动，此多项式对实际$f(x)$的逼近效果会很差。

使用函数demoWiggle（此处未列出，但包含在NMM工具箱中）可以从表格中选择一些合适的点来创建一系列高阶多项式。在每种情况中，台阶尽可能置于所选数据的中间。图10-12包含了阶数从2到9的一系列插值多项式的图。虽然插值多项式（实线）通过了支点，但是它们在支点之间都明显地偏离了实际函数$f(x)$的基本趋势，且各插值函数的振荡程度随多项式阶数的增加而变强。

图10-12中每个图的虚线表示插值多项式的外插函数。即使是用低阶的多项式，外插对$f(x)$的逼近也非常不准确。高阶多项式在支点间有摆动，在支点外它们比低阶多项式偏离得更迅速。 553

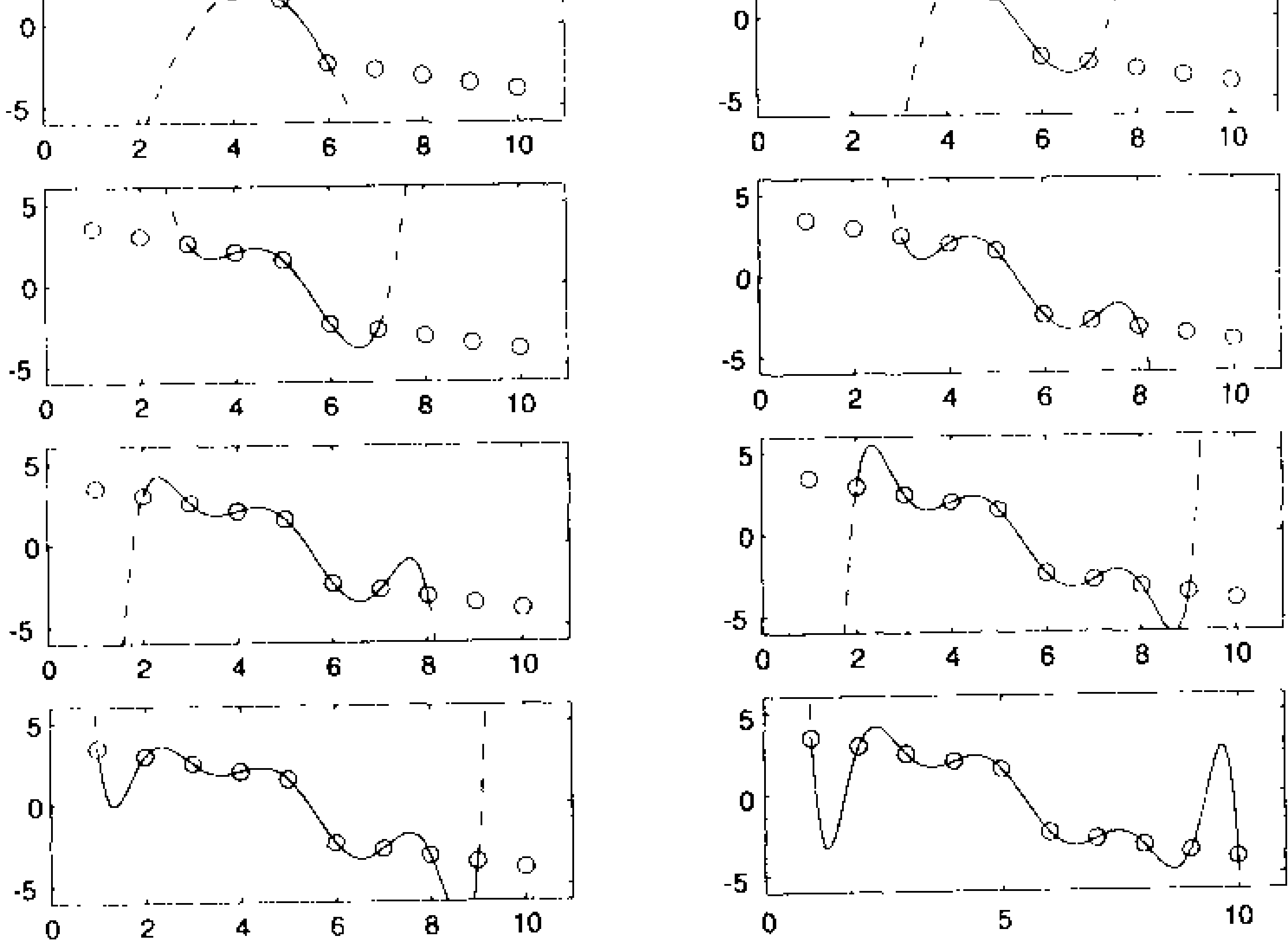


图10-12 多项式摆动和外插随插值多项式阶数的增加而变化的情况，空心圆表示给定数据，实线为插值函数，虚线为外插函数

10.3 分段多项式插值

分段多项式插值为高阶多项式插值的不足提供了一种实用的解决方案。它不是由一个穿过大量支点的插值式来对函数逼近，而是采用一个低阶插值式的集合来进行插值，每一个插值式都定义在整个域的子区间上。相邻分段插值式的结合处被称为分界点（*breakpoint*）或节点（*knot*）。

分段函数给插值带来了新的特征。邻接分段函数之间的关系（如果有的话）是最重要的关系。另外，邻接插值式的形状会受插值式及其导数在节点处连续性的约束。图10-13描述了一般的连续性约束。顶上的图形描述了在节点处连续的分段线性函数；中间图形描述了两个 554

邻接的二次插值式，它们也在节点处连续。如果说分段线性函数斜率不连续还可以理解，那么二阶插值式斜率不连续就不能反映出数据的趋势了。最下面的图形中，两个邻接三阶多项式在节点处的一阶和二阶导数都连续，此插值式称为三阶样条，这是10.3.5节的主题。

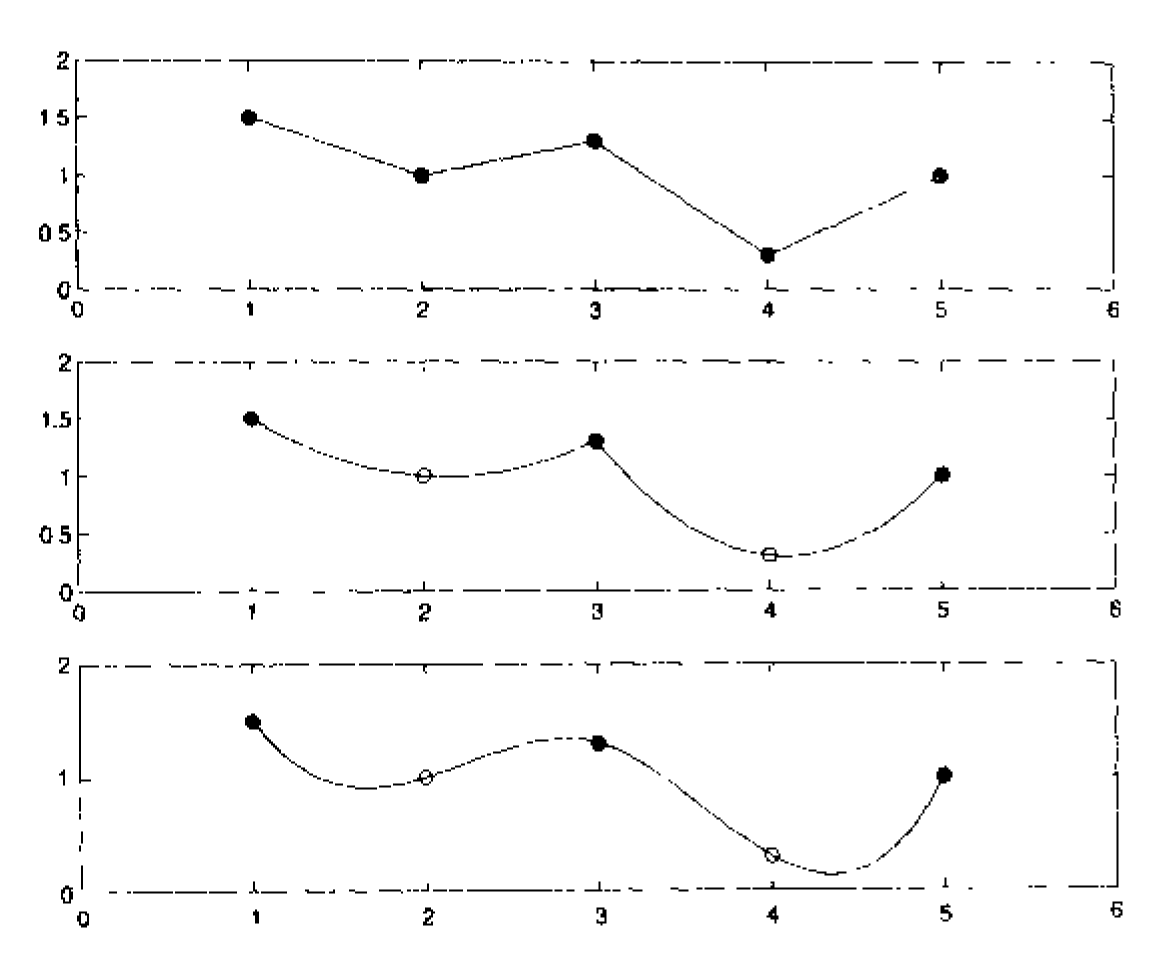


图10-13　分段多项式插值穿过相同的五个支点的连续性条件，实心点为分界点。顶图描述了分段线性插值，中间图形描述了节点处斜率不连续的分段二阶插值，下图描述了节点处一阶二阶导数都连续的分段三阶插值

符号变更　在本章剩余的部分中，将用$P_i(x)$表示定义在分段连续曲线的第i段上的多项式，第i段由$x_i \leqslant x \leqslant x_{i+1}$来界定。

计算任务　对任意阶数的插值多项式，在用分段插值多项式时需要构造并计算插值式。因为分段插值式不是全局的（即不是定义在所有支点上的单一函数），在计算插值函数之前必须选择合适的子区间。一般地，在点$\hat{x}$进行分段插值所需的任务是：

555

1. 在支点集(x_i, y_i)上定位$\hat{x}$，$i=1,\cdots,n$。
2. 计算在相关支点上定义的局部插值多项式的系数。
3. 求出局部插值多项式。

如果分段插值式一阶或更高阶的导数连续，那么邻接插值式可能会耦合。这样，虽然单个插值式是局部的，但由于在节点处连接，它们也会显露出一些全局的性质。这就是三阶样条插值式。

10.3.1　分段线性插值

最简单的分段插值方案是在每对连续的节点间使用线性函数进行插值。这实际上就是MATLAB的`plot`函数在用实线线段来画“曲线”时选取屏幕像素（或打印页上的光栅图像点）所用的方法。下面的两条MATLAB语句

```
>> x = linspace(0,2*pi);
>> plot(x,sin(x))
```

产生一个描述正弦函数的平滑曲线。因为使用了大量的分界点，插值式中斜率的不连续性就不太明显了。但如果用`x=linspace(0,2*pi,10)`重复画图，斜率的不连续性就容易看出来了。

考虑用插值方法来求出y在$x = 0.75$时的值，函数y由下列数据定义：

x_i	0.2	0.6	1.0	1.4
y_i	0.5535	1.0173	1.0389	0.8911

相关的支点是$(x,y) = (0.6,1.0173)$和$(x,y) = (1.0,1.0389)$。使用拉格朗日基本插值公式函数（参考方程（10-11））我们得到：

$$L_1(0.75)=\frac{0.75-1.0}{0.6-1.0}=0.6250 \quad \text{和} \quad L_2(0.75)=\frac{0.75-0.6}{1.0-0.6}=0.3750$$

$$P_1(0.75)=1.0173L_1(0.75)+1.0398L_2(0.75)=1.0254$$

556

分段线性插值的实现中惟一复杂的地方就是在构造插值式时要选用合适的支点对。在一个对任意数据集进行分段线性插值的MATLAB函数中，支点的查找和插值函数的计算都需要自动完成。

10.3.2 查找支点

已知表格数据(x_i,y_i)，$i=1,\cdots,n$，我们选择一个包含$\hat{x}$的x值的子集来构造分段多项式插值。对分段线性插值来说，只需要两个支点，即x_{left}和x_{right}，使$x_{left} \leqslant \hat{x} \leqslant x_{right}$。这里将讨论两个选择算法：渐进搜索法（增量查找法）和折半搜索法（二分查找法）。两种算法中，都假设数据集中的x是单调递增的。介绍完折半搜索后，还将简单描述处理非单调递增x值数据集的程序。

渐进搜索 渐进搜索是求有序向量中某元素下标的一种强制策略，这个有序向量包含一个测试值。如果向量`x`的元素是单调递增的，那么下面的代码段可用来求出使`x(i)` ≤ `xhat<x(i+1)`成立的下标`i`（查找过程在`while`循环语句中）：

```
n = length(x);
if xhat<x(1) | xhat>x(n)
   error(sprintf('Test value of %g is not in range of x',xhat));
end
i = n;                      %  Search backward from n to 1
while x(i)>xhat & i>1
  i = i - 1;
end
```

范围测试`xhat<x(1)|xhat>x(n)`是用来提醒用户输入数据有错误[⊖]的防错性程序设计。对于不常用到的短向量查找，渐进算法就足够了，而对长`x`向量来说，这个算法就比较慢了，因为：第一，代码不是向量化的；第二，算法效率低。求区间下标时，下节介绍的折半搜索算法通常比渐进搜索算法使用的操作更少，但是它不能向量化。

557

在例3.12中，介绍了渐进搜索算法中`while`循环的向量化表示。对`while`循环逻辑的直接转换就是：

```
i = max( find(x<=xhat) );    %  Find largest i such that x(i)<= xhat
i = min( i, length(x)-1 );   %  Fix case where i=length(x)
```

⊖ 要使用渐进搜索算法对一个数据集进行外插，必须将此检查去掉，另外还需要增加一些逻辑来选择合适的支点下标。

向量化的渐进搜索代码仍然不如折半搜索算法快。虽然向量化加速了计算，但是在数据集中从i搜索到n时比较运算太多。折半搜索在查找区间中的某个值时所使用的操作较少。

折半搜索　折半搜索不是从支点的一端搜索到另一端，而是使用二分算法（参照6.4节）。程序清单10-6中的binSearch函数在单调递增的x值向量中寻找一个测试值xhat。本章剩余部分的分段多项式插值程序都使用折半搜索。

程序清单10-6　函数binSearch用折半搜索算法在单调递增值中寻找一个元素下标

```
function ia = binSearch(x,xhat)
% binSearch  Binary search to find index i such that x(i)<= xhat <= x(i+1)
%
% Synopsis:  i = binSearch(x,xhat)
%
% Input:     x    = vector of monotonic data
%            xhat = test value
%
% Output:    i = index in x vector such that x(i)<= xhat <= x(i+1)

n = length(x);
if xhat<x(1) | xhat>x(n)
   error(sprintf('Test value of %g is not in range of x',xhat));
end

ia = 1;  ib = n;          %  Initialize lower and upper limits
while ib-ia>1
  im = fix((ia+ib)/2);    %  Integer value of midpoint
  if x(im) < xhat
    ia = im;              %  Replace lower bracket
  else
    ib = im;              %  Replace upper bracket
  end
end                       %  When while test is true, ia is desired index
```

558

查找前的排序　渐进搜索和折半搜索算法经过修改可以处理单调递减的x数据($x_n>x_{n-1}>...>x_2>x_1$)。更简单的方法是在插值进行之前就用fliplr或flipud函数对数据进行翻转拷贝。如果x和y是单调递减的行向量或列向量，下面的语句就产生xx和yy列向量，其中xx(i)是单调递增的：

```
>> xx = flipud(x(:));    %  x(:) is always a column vector
>> yy = flipud(y(:));
```

然后再用定义支点的xx和yy向量进行插值（为什么不只翻转x?）。

如果数据集是无序的，就需要在搜索和插值之前进行排序。使用内置sort函数可以很简单地完成这一任务。为保持x和y之间的一致性，需要使用两个参数的sort版本：

```
>> [xs,is] = sort(x);    %  xs is sorted x; "is" is the sorting permutation
>> ys = y(is)            %  ys is y rearraged  to be consistent with xs
```

然后使用定义支点的xs和ys进行插值。

10.3.3　linterp函数

程序清单10-7中的linterp函数对存储在向量x和y中的数据进行分段线性插值，支点使

用binSearch函数搜索，插值式由线性拉格朗日基本函数求出（参照10.2.2节）。

函数linterp是分段线性插值的直接实现。内置函数interp1实现了几种类型的插值，包括分段线性插值。函数interp1比函数linterp做的工作多得多，因此它的代码更复杂，线性插值的简单性在其中也就显现不出来了。

程序清单10-7 分段线性插值函数linterp

```
function yi = linterp(x,y,xi)
% linterp   Piecewise linear interpolation in a table of (x,y) data
%
% Synposis:  yi = linterp(x,y,xi)
%
% Input:     x,y = vectors containing the tabulated data
%            xi  = value of x at which function y = f(x) is desired
%
% Output:    yi = value of y at xi obtained by linear interpolation

i = binSearch(x,xi);                      % Find appropriate data pair
L1 = (x(i+1) - xi)/(x(i+1) - x(i));       % Evaluate basis functions
L2 = (xi - x(i))/(x(i+1) - x(i));
yi = y(i)*L1 + y(i+1)*L2;                 % Evaluate interpolant
```

例10.12 热电偶数据的线性插值

例9.12中，J型热电偶的校准值被拟合为线性、二阶和三阶多项式。曲线拟合的结果是一个单独函数，它在整个校准数据上根据emf推算出。在本例中我们将使用分段线性插值根据emf推算出温度值，以及在相同数据上根据温度推算出emf值。图9-12描述了此数据的图形。 [559]

下面的语句将数据读入MATLAB变量，并使用linterp函数求出emf为0.8mV时的温度值：

```
>> [v,t] = loadColData('Jtcouple.dat',2,1,3);
>> linterp(v,t,0.8)
ans =
    15.7227
```

函数linterp可以同样容易地对T的函数emf进行插值。由刚刚定义的emf和T向量，可得温度37°C下的emf值为：

```
>> linterp(t,v,37)
ans =
    1.9024
```

10.3.4 分段三阶Hermite插值

按逻辑上说，介绍完分段线性插值之后，紧跟着应该介绍分段二阶插值，但实际上，分段二阶函数并不一定比分段线性函数更好。对已知数据进行插值，分段三阶函数更有用。在本节和下节，会介绍两种不同类型的分段三阶插值函数：Hermite多项式和三阶样条函数。 [560]

在前面所有的推导中，n阶多项式定义时都要求它通过$n+1$个指定支点。而Hermite多项式（*Hermite polynomial*）则要求在每个支点处的函数及其一阶导数一致，因此产生了两个理想的属性：

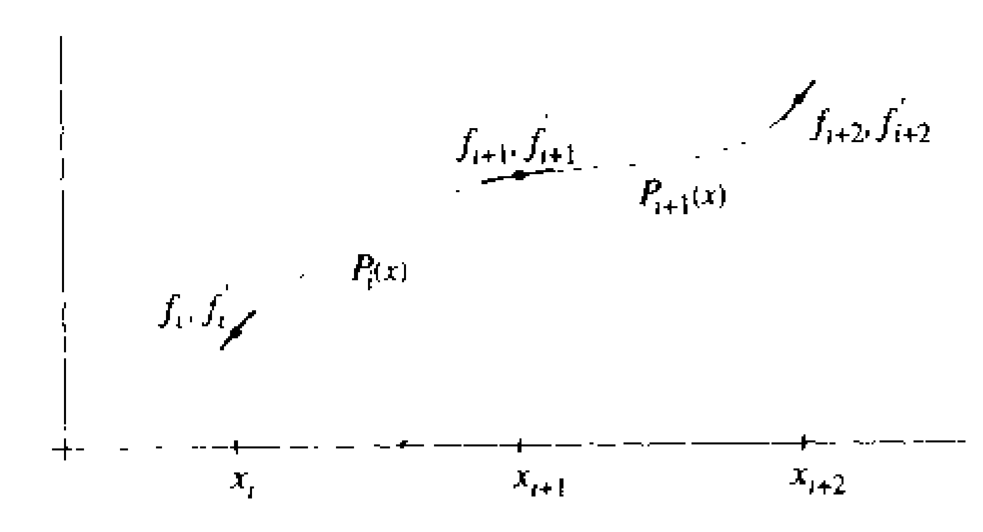


图10-14 分段三阶Hermite插值多项式由节点处的函数值和斜率决定。$P_i(x)$是匹配$f(x_i)$、$f'(x_i)$、$f(x_{i+1})$和$f'(x_{i+1})$的三阶函数

- 分段Hermite多项式的各区间在每个支点的一阶导数连续。
- 插值函数的形状可以匹配得更好，因为在支点处函数的正切与Hermite多项式的正切一致。

因为插值Hermite多项式需要求解$y=f(x)$和$f'(x)$的值，所以它对解析函数插值比对离散数据（如实验数据）插值更适合。除非绝对必要，一般并不推荐对离散数据求$f'(x)$。对于$f'(x)$未知的离散数据，推荐使用样条插值（参见10.3.5节）。

图10-14分别描述了定义在闭区间$[x_i,x_{i+1}]$和$[x_{i+1},x_{i+2}]$上的两个分段多项式$P_i(x)$和$P_{i+1}(x)$。$P_i(x)$的三阶形式可写为：

$$P_i(\hat{x})=a_i+b_i(\hat{x}-x_i)+c_i(\hat{x}-x_i)^2+d_i(\hat{x}-x_i)^3 \tag{10-32}$$

其中a_i、b_i、c_i和d_i是第i个片段的系数，$\hat{x}$是计算Hermite多项式[⊖]时的x取值。在MATLAB程序中，将方程（10-32）写成数学意义相同、但是计算上更有效的嵌套形式（另见例3.7）：

561

$$P_i(\hat{x})=a_i+(\hat{x}-x_i)(b_i+(\hat{x}-x_i)(c_i+(\hat{x}-x_i)d_i)) \tag{10-33}$$

方程（10-32）和方程（10-33）中的未知系数a_i、b_i、c_i和d_i要求满足下面等式：

$$P_i(x_i)=f(x_i),\quad P_i'(x_i)=f'(x_i),\quad P_i(x_{i+1})=f(x_{i+1}),\quad P_i'(x_{i+1})=f'(x_{i+1}) \tag{10-34}$$

虽然$P_i(x)$和$P_{i+1}(x)$在x_{i+1}处匹配，但是前面的四个条件足以不依赖于a_{i+1}、b_{i+1}、c_{i+1}和d_{i+1}的值来定义a_i、b_i、c_i和d_i。由于$f(x)$和$f'(x)$惟一，因此可以保证P_i和P_i'的连续性。

求解出$P_i(x)$的系数 均差可以用来定义一个插值多项式的系数，其中函数及其导数在支点上已知。满足方程（10-34）的三阶多项式可写为（可参考Conte和de Boor[11,2.7节]）：

$$\begin{aligned}P(x)=f(x_i)&+f[x_i,x_i](\hat{x}-x_i)\\&+f[x_i,x_i,x_{i+1}](\hat{x}-x_i)^2\\&+f[x_i,x_i,x_{i+1},x_{i+1}](\hat{x}-x_i)^2(\hat{x}-x_{i+1})\end{aligned} \tag{10-35}$$

其中，由定义知

$$f[x_i,x_i]=f'(x_i)\quad 和\quad f[x_{i+1},x_{i+1}]=f'(x_{i+1})$$

且

⊖ P_i是定义在第i个片段上的多项式，$P_i(x)$必须是一个阶多项式。

$$f[x_i,x_i,x_{i+1}]=\frac{f[x_i,x_{i+1}]-f[x_i,x_i]}{x_{i+1}-x_i}$$

$$f[x_i,x_{i+1},x_{i+1}]=\frac{f[x_{i+1},x_{i+1}]-f[x_{i+1},x_i]}{x_{i+1}-x_i}$$

$$f[x_i,x_i,x_{i+1},x_{i+1}]=\frac{f[x_i,x_{i+1},x_{i+1}]-f[x_i,x_i,x_{i+1}]}{x_{i+1}-x_i}$$

方程（10-35）中的三次项包含因式$(\hat{x}-x_{i+1})$，它不是方程（10-32）所期望的形式。将$(\hat{x}-x_{i+1})=(\hat{x}-x_i)+(x_i-x_{i+1})$代入方程（10-35）得： [562]

$$\begin{aligned}P_i(x)=f(x_i)&+f[x_i,x_i](\hat{x}-x_i)\\&+\left\{f[x_i,x_i,x_{i+1}]-f[x_i,x_i,x_{i+1},x_{i+1}](x_{i+1}-x_i)\right\}(\hat{x}-x_i)^2\\&+f[x_i,x_i,x_{i+1},x_{i+1}](\hat{x}-x_i)^3\end{aligned}\tag{10-36}$$

将方程（10-36）与方程（10-32）比较可知a_i、b_i、c_i和d_i是$(\hat{x}-x_i)$项的系数。将这些系数进行直接的代数简化，得

$$\begin{aligned}a_i&=f(x_i),\\b_i&=f'(x_i),\\c_i&=\frac{3f[x_i,x_{i+1}]-2f'(x_i)-f'(x_{i+1})}{(x_{i+1}-x_i)}\\d_i&=\frac{f'(x_i)-2f[x_i,x_{i+1}]+f'(x_{i+1})}{(x_{i+1}-x_i)^2}\end{aligned}\tag{10-37}$$

实现 使用三阶Hermite多项式进行插值主要有三个步骤。首先，计算出所有分段三阶片段表示的系数。然后，对每个输入x值定位其合适的片段。最后，计算插值式。计算包含在程序清单10-8中的hermint函数中，此函数的代码在介绍了输入输出参数之后讨论。

函数hermint的调用形式为：

```
yhat = hermint(x,f,fp,xhat)
```

对向量x的每个元素，向量f和fp分别给出$y=f(x)$和$f'(x)$。输入参数xhat是在该处要计算插值的x值的标量或向量；输出yhat是一个包含每个xhat元素插值结果的标量或向量。

hermint函数中的代码分为三块。第一块按照方程（10-37）构造分段三阶插值式的系数。为保证向量化计算中矩阵的兼容性，所有的输入变量都要转变成列向量，否则，计算就会过于简单。

hermint函数中的第二和第三块代码用来求解出期望的$\hat{x}$值处的插值式。为完成这个任务，需要选择合适的分段。图10-15描述了标量$\hat{x}$在由六个节点定义的分段插值式上的位置。对输入向量xhat的每一个元素，使用折半搜索来寻找使x(i) <xhat(m) <x(i+1)成立的下标i。考虑到xhat值是一个向量，下标i也保存在向量中。

即使只需要计算一个yhat值，仍需构造所有的三阶片段。这虽然有些低效，但这样做也有优势，体现在程序代码逻辑简单，且在xhat为向量时通过已向量化的代码可取得最大效率。函数hermint适合在所有输入x值的范围上计算插值式。为更高效地计算xhat的值为标量的情况，在调用hermint函数时参数x、f和fp的长度应该为2，这可能需要在调用hermint函数之前调用binSearch函数。 [563]

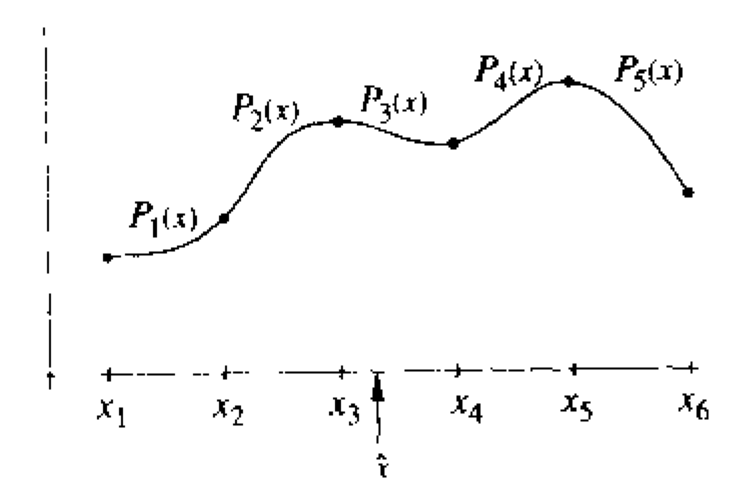


图10-15 在计算插值 $y(\hat{x})$ 时合适的片段分布。图中描述了由6个节点产生的插值式。$y(\hat{x})$ 的值将由 $P_3(x)$ 求出

程序清单10-8 函数hermint进行分段三阶Hermite插值

```
function yhat = hermint(x,f,fp,xhat)
% hermint  Piecewise-cubic Hermite interpolation
%
% Synopsis: yhat = hermint(x,f,fp,xhat)
%
% Input:     x      = vector of independent variable values
%            f, fp = vectors of f(x) and f'(x)
%            xhat = (scalar or vector) x values where interpolant is evaluated
%
% Output:    yhat = scalar or vector value of cubic hermite interpolant at
%                   x = xhat.  size(yhat) = size(xhat)

n = length(x);
if      length(f)~=n,    error('x and f are not compatible');
elseif length(fp)~=n,    error('x and fp are not compatible');   end

% --- Construct coefficients of the piecewise interpolants
x = x(:),   xhat = xhat(:);     % Convert to column vectors
f = f(:),   fp = fp(:),
dx = diff(x);                   % Vector of x(i+1) - x(i) values
divdif = diff(f)./dx;           % Vector of divided differences, f[x(i),x(i+1)]
a = f(1:n-1);
b = fp(1:n-1);
c = ( 3*divdif - 2*fp(1:n-1) - fp(2:n) ) ./dx;
d = ( fp(1:n-1) - 2*divdif + fp(2:n) )  ./dx.^2;

% --- Locate each xhat value in the x vector
i = zeros(size(xhat));  %  i is index into x such that x(i) <= xhat <= x(i+1)
for m=1:length(xhat)    %  For vector xhat: x( i(m) ) <= xhat(m) <= x( i(m)+1 )
  i(m) = binSearch(x,xhat(m));
end

% --- Vectorized evaluation of the piecewise polynomials
xx = xhat - x(i);
yhat = a(i) + xx.*(b(i) + xx *(c(i) + xx.*d(i)) );
```

564

已知分段多项式片段的系数和对应于输入*xhat*值的片段的合适下标i，我们可用两条向量化的语句来计算插值式：

```
xx = xhat - x(i);
yhat = a(i) + xx.*(b(i) + xx.*(c(i) + xx.*d(i)) );
```

变量xx是与xhat长度相同的向量。不论x的长度为多少，子表达式x(i)的结果都是长度为i的向量。考虑下列MATLAB交互语句：

```
>> x = 0:10;                  % row vector of independent x values
>> xhat = [3.2  5.4  7.1];    % interpolate at each xhat
>> i = zeros(size(xhat))      % pre-allocate the index vector
i =
     0     0     0
```

现在，使用binSearch函数求小于每个xhat的x中最大元素的下标：

```
>> for k=1:length(xhat),  i(k) = binSearch(x,xhat(k));  end
>> i
i =
     4     6     8
```

与这些下标对应的x元素为：

```
>> x(i)
ans =
     3     5     7
```

因此，语句xhat - x(i)返回一个向量，

```
>> xx = xhat-x(i)
xx =
    0.2000    0.4000    0.1000
```

注意，length(x) = 11。

565

例10.13 Hermite插值的演示

程序清单10-9 函数demoHermite使用hermint函数对$y = xe^{-x}$进行分段三阶Hermite插值逼近并画图

```
function demoHermite(n)
% demoHermite  Cubic Hermite interpolation of y = x*exp(-x) for 0 <= x <= 8
%
% Synopsis:    demoHermite
%              demoHermite(n)
%
% Input:       n = (optional) number of knots used to define the interpolant
%                  Default:  n = 4
%
% Output:      Plot of cubic Hermite approximation to y = x * exp(-x)

if nargin<1,  n = 4;  end

x = linspace(0,8,n);                  %  vector of knots
y = x.*exp(-x);                       %  f(x)
yp = (1 - x).*exp(-x);                %  f'(x)
xi = linspace(min(x),max(x));         %  Evaluate interpolant at xi
ye = xi.*exp(-xi);                    %  Exact f(x) for comparison

yi = hermint(x,y,yp,xi);
err = norm(yi-ye(:));
fprintf('error = %12.2e  with %d knots\n',err,n);
```

```
plot(x,y,'bo',xi,ye,'b-',xi,yi,'r--');
legend('Given','x*exp(-x)','Hermite');
text(4,0.2,sprintf('%d knots',n));
axis([0 8 0 0.5]);
```

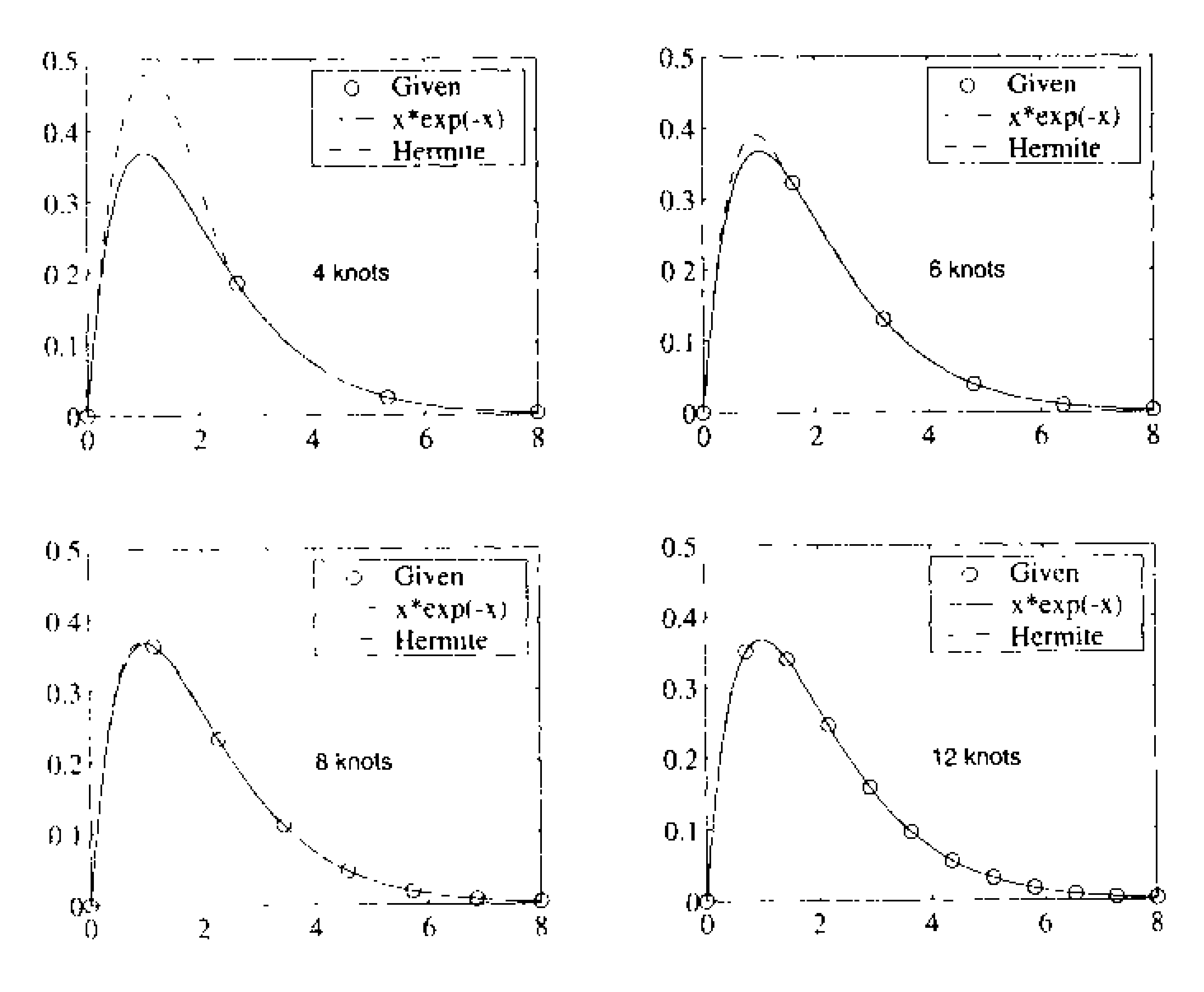
566

程序清单10-9中的demoHermite函数在$0 \leqslant x \leqslant 8$上产生对函数$y=xe^{-x}$的分段三阶Hermit插值逼近。它只有惟一的输入参数n，表示在x范围中使用的节点数，这样三阶片段的总数就是$n-1$。运行demoHermite得到一个三阶Hermite插值式与原始数据的比较图。计算绝对误差$E=\|\hat{y}-y_e\|_2$，其中$\hat{y}$为插值式的值，y_e为输入函数的精确值。$\hat{y}$和y_e都在$0 \leqslant x \leqslant 8$取值范围内的100个点上计算。

图10-16描述了demoHermite产生的四个图，它们是按照n值的大小排列的。随着n的增大，插值式会很快地接近原始函数。误差随节点数量大小的变化总结如下：

节点	$\|\hat{y}-y_e\|_2$
4	4.2×10^{-1}
8	2.2×10^{-2}
16	1.2×10^{-3}
32	6.4×10^{-5}
64	3.8×10^{-6}

插值式中的误差一般随着节点数的增加而减小。就像所有类型的分段多项式插值一样，这里也没有多项式摆动（比较10.2.4节）。



567

图10-16　对函数$y=xe^{-x}$的分段三阶Hermite插值逼近

10.3.5 三阶样条插值

虽然分段三阶Hermite插值可产生平滑变化的插值式，但它有一个明显的缺点，就是在每个分界点处输入函数的斜率必须已知。而像从实验中获得的数据，这个斜率就不存在。三阶样条插值可以解决这个问题，同时能保持插值式所期望的平滑度。

图10-17描述了通过n个分界点的三阶样条插值式。每个$P_i(x)$是一个三阶多项式，且在每个分界点处$P_i(x)$、$P_i'(x)$和$P_i''(x)$是连续的。由于$P_i''(x)$连续，相邻的分段三阶多项式就可以很好地耦合（即样条曲线在该点平滑、曲率变化均匀，译者注）。结果，所有$P_i(x)$系数必须同步计算，这就得出n个分界点上含$n-2$个方程的方程组。因此，虽然三阶样条避免了在分界点处指定$f'(x)$，但是三阶样条插值式的求解代价很大。幸好，在很多应用中，这种代价不是主要的问题。

三阶样条的定义方程 三阶样条的定义方程可由分段三阶Hermite多项式得到。三阶样条的每个$P_i(x)$都形如方程（10-32），此处重写为：

$$P_i(\hat{x}) = a_i + b_i(\hat{x} - x_i) + c_i(\hat{x} - x_i)^2 + d_i(\hat{x} - x_i)^3 \tag{10-38}$$

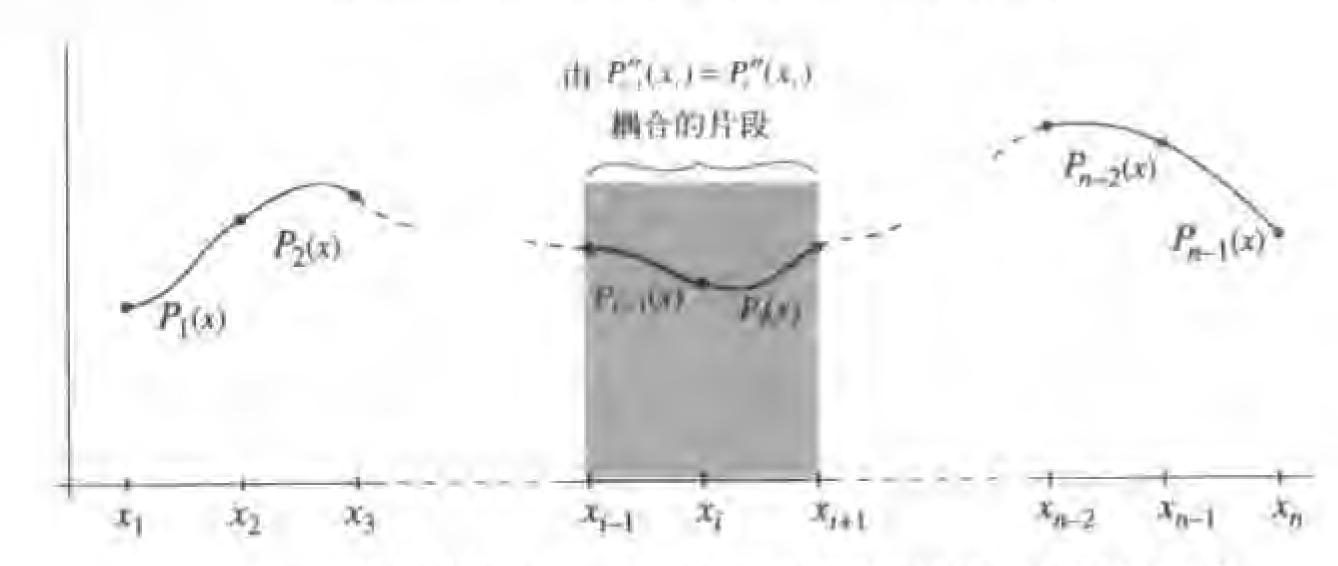


图10-17 三阶样条插值函数。$P_i''(x)$的连续性可使邻接分段多项式耦合

568

三阶Hermite和三阶样条插值多项式都有连续的函数值及一阶导数（即$P_{i-1}(x_i) = P_i(x_i)$，$P'_{i-1}(x_i) = P_i'(x_i)$），只是分界点一阶导数的值不同。对三阶Hermite多项式来说，$P_i'(x_i)$的值由输入数据给定，即$P_i'(x_i) = f'(x_i)$。对三阶样条来说，$P_i'(x_i)$的值通过迫使$P_i''(x_i)$在分界点连续而得到。

除了$f'(x)$的值未知外，方程（10-37）中计算a_i，b_i，c_i和d_i的公式也可应用在三阶样条插值上。注意，既然

$$P_i'(\hat{x}) = b_i + 2c_i(\hat{x} - x_i) + 3d_i(\hat{x} - x_i)^2$$

且

$$P_i'(x_i) = b_i, \qquad P_{i-1}'(x_i) = b_{i-1}$$

不失一般性地，我们可以用b_i来代替方程（10-37）中的$f'(x_i)$，用b_{i-1}来代替其中的$f'(x_{i-1})$。对第$i-1$个片段，有：

$$\begin{aligned} a_{i-1} &= f(x_{i-1}) \\ b_{i-1} &= ???, \\ c_{i-1} &= \frac{3f[x_{i-1}, x_i] - 2b_{i-1} - b_i}{(x_i - x_{i-1})} \\ d_{i-1} &= \frac{b_{i-1} - 2f[x_{i-1}, x_i] + b_i}{(x_i - x_{i-1})^2} \end{aligned} \tag{10-39}$$

b_i未知，现在的目标就是推导一个用来计算b_i的过程⊖。

决定b_i的条件是要求样条片段在分界点处的二阶导数必须连续。由方程（10-38），得

$$P_i''(\hat{x}) = 2c_i + 6d_i(\hat{x}-x_i)$$

因此由 $P_{i-1}''(x_i) = P_i''(x_i)$ 可知，

$$2c_{i-1} + 6d_{i-1}\Delta x_{i-1} = 2c_i$$

[569] 其中$\Delta x_{i-1}=x_i - x_{i-1}$。代入方程（10-39）中的$c_{i-1}$和$d_{i-1}$表达式并简化，得

$$\begin{aligned}\Delta x_i b_{i-1} + 2(\Delta x_i + \Delta x_{i-1})b_i + \Delta x_{i-1} b_{i+1} \\ = 3(f[x_i,x_{i+1}]\Delta x_{i-1} + f[x_{i-1},x_i]\Delta x_i)\end{aligned} \qquad (10\text{-}40)$$

方程右边已知，左边有未知的b_{i-1}、b_i和b_{i+1}。由方程（10-40）可知，需要由$P_i''(x)$的连续性将邻接片段的样条系数耦合起来。图10-17从图形上描述了这种耦合。既然b_i耦合，要求出三阶样条插值多项式的系数，就需要求解一些联立方程组。一旦b_i已知，剩下的系数a_i、c_i和d_i就可以由方程（10-39）计算出来。

定义如下辅助变量可使关于b_i的方程组更容易操作：

$$\alpha_i = \Delta x_i \qquad (10\text{-}41)$$

$$\beta_i = 2(\Delta x_i + \Delta x_{i-1}) = 2(\alpha_i + \gamma_i) \qquad (10\text{-}42)$$

$$\gamma_i = \Delta x_{i-1} \qquad (10\text{-}43)$$

$$\delta_i = 3(f[x_i,x_{i+1}]\Delta x_{i-1} + f[x_{i-1},x_i]\Delta x_i) \qquad (10\text{-}44)$$

方程（10-40）变成

$$\alpha_i b_{i-1} + \beta_i b_i + \gamma_i b_{i+1} = \delta_i \qquad (10\text{-}45)$$

这里有几个方程？

图10-17描述了穿过n个已知数据对(x_i,y_i)，$i = 1,\ldots,n$的三阶样条。这里有$n-1$个分段三阶多项式，既然每个多项式有四个系数，那么整个的样条就需要确定$4(n-1)$个参数。同时也需要同样多的约束条件。下面的约束条件就是来自于方程（10-45）的。

参数个数	约束条件类型
$2(n-1)$	$n-1$个多项式都要在其分界点上匹配2个给定的y值
$n-2$	$P'_{i-1}(x_i)=P'_i(x_i)$的内部节点数
$n-2$	$P''_{i-1}(x_i)=P''_i(x_i)$的内部节点数
$4n-6$	总约束条件个数

于是，为了求解用来构成样条的$n-1$个分段三阶多项式，还需要两个附加条件来完全确定$4(n-1)$个参数。这些条件可以在图10-17的辅助下推导。

我们已经知道，未知的b_i是分界点处的斜率值。$b_2,b_3,\ldots,b_{n-1}$是$n-2$个未知的内部斜率，而[570] b_1和b_n是样条端点的斜率⊖。所以，未知条件的一个解释就是：必须指定端点斜率值b_1和b_n，这样才能计算出$n-2$个内部b_i来。更一般地，两个端点条件（*end condition*，又称边界条件）

⊖ 很多人用c_i（就是节点的二阶导数）来推导三阶样条的方程。这里的推导参照了de Boor[14]。

⊖ 在计算 $P_{n-1}(\hat{x})$时不需要b_n，但是b_n的值在计算b_{n-1}时要用到。将方程（10-45）中的i设为$n-1$就可以看出。

必须独立规定，这样才能惟一确定样条。在下节中，将对三种端点条件进行推导：

1. 固定斜率端点条件
2. “自然”样条端点条件
3. “非节点”端点条件

在介绍端点条件之前，将方程（10-45）写成矩阵的形式会很有用。

$$\begin{bmatrix} \beta_1 & \gamma & 0 & & & \\ \alpha_2 & \beta_2 & \gamma_2 & 0 & & \\ 0 & \alpha_3 & \beta_3 & \gamma_3 & 0 & \\ & 0 & \ddots & \ddots & \ddots & 0 \\ & & 0 & \alpha_{n-1} & \beta_{n-1} & \beta_{n-1} \\ & & & 0 & \alpha_n & \beta_n \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \\ \vdots \\ \delta_{n-1} \\ \delta_n \end{bmatrix} \tag{10-46}$$

这是关于未知b_i的三对角线性方程组。如上所示，这里有n个方程，每个方程有n个未知数。为方便端点条件的计算，其中两个额外方程——b_1和b_n的方程——已用MATLAB代码表示出来。一旦方程组(10-46)中的b_i计算出来，剩下的系数就可以用方程（10-39）来计算，即

$$a_i = y_i \tag{10-47}$$

$$c_i = \frac{3f[x_i, x_{i+1}] - 2b_i - b_{i+1}}{(x_{i+1} - x_i)} \tag{10-48}$$

$$d_i = \frac{b_i - 2f[x_i, x_{i+1}] + b_{i+1}}{(x_{i+1} - x_i)^2} \tag{10-49}$$

到目前为止，计算三阶样条系数的基本算法框架已经完成了。已知数据集(x_i, y_i)，$i = 1, \cdots, n$和合适的端点条件，就可以由如下步骤得到样条系数。

1. 建立方程组（10-46）
2. 解此方程组得出系数向量b
3. 用方程（10-47）到方程（10-49）来计算样条的剩余系数（$i = 1, \ldots, n-1$）

571

已知系数向量a、b、c和d，样条插值式在点$\hat{x}$的值可由以下步骤得到：

1. 确定使$x_i \leqslant \hat{x} \leqslant x_{i+1}$的下标$i$
2. 计算方程（10-38）

固定斜率端点条件 三阶样条端点处的斜率分别是b_1和b_n。整理方程（10-46）并使这两个值固定。三对角方程组（10-46）中第一和最后一个方程是：

$$\beta_1 b_1 + \gamma_1 b_2 = \delta_1$$
$$\alpha_{n-1} b_{n-1} + \beta_n b_n = \delta_n$$

令$\beta_1 = 1$，$\gamma_1 = 0$，$\delta_1 = f'(x_1)$、$\alpha_n = 0$，$\beta_n = 1$，$\delta_n = f'(x_n)$，则关于b_1和b_n的方程可简化为：

$$b_1 = f'(x_1)$$
$$b_n = f'(x_n)$$

例10.4演示了这些公式的使用。在下一节中，将固定斜率端点条件合并到MATLAB程序中，用它来计算三阶样条系数。

例10.14 样条系数的手工计算

求穿过$(x,y)=(1,1)$、$(2,3)$、$(3,2)$和$(4,4)$，且$x=1$和$x=4$处的斜率为零的三阶样条（计算之前，最好先画出样条的大略形状）。

对$i=2$、3直接计算方程（10-41）到方程（10-44），得

$$\alpha_2 = x_3 - x_2 = 1$$
$$\gamma_2 = x_2 - x_1 = 1$$
$$\beta_2 = 2(\alpha_2 + \gamma_2) = 4$$
$$\delta_2 = 3\left[\frac{y_3 - y_2}{x_3 - x_2}(x_2 - x_1) + \frac{y_2 - y_1}{x_2 - x_1}(x_3 - x_2)\right] = 3$$
$$\alpha_3 = x_4 - x_3 = 1$$
$$\gamma_3 = x_3 - x_2 = 1$$

572

$$\beta_3 = 2(\alpha_3 + \gamma_3) = 4$$
$$\delta_3 = 3\left[\frac{y_4 - y_3}{x_4 - x_3}(x_3 - x_2) + \frac{y_3 - y_2}{x_3 - x_2}(x_4 - x_3)\right] = 3$$

由这些得到的系数以及前面提到的b_1和b_n的方程，关于b_i的方程组变成:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 3 \\ 0 \end{bmatrix}$$

手工计算这个方程组时，可以用关于b_1和b_4的平凡方程来从第二和第三个方程中消去b_1和b_4:

$$\begin{bmatrix} 4 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 3 - b_1 \\ 3 - b_4 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

解此2×2方程组得$b_2=3/5$、$b_3=3/5$。另外，MATLAB中对整个4×4方程组的解法为:

```
>> A = [1 0 0 0; 1 4 1 0; 0 1 4 1; 0 0 0 1];
>> d = [0; 3; 3; 0];
>> b = A\d
b =
         0
    0.6000
    0.6000
         0
```

解出b后，可由方程（10-47）到方程（10-49）计算出剩下的样条系数，结果为

$$a = \begin{bmatrix} 1 \\ 3 \\ 2 \end{bmatrix}, \quad c = \begin{bmatrix} 5.4 \\ -4.8 \\ 4.8 \end{bmatrix}, \quad d = \begin{bmatrix} -3.4 \\ 3.2 \\ -3.4 \end{bmatrix}$$

注意，这里a_4，c_4和d_4并没有计算出来，因为对四个输入数据对只定义了三个三阶片段。使用b_4的值只是为了使x_4处的零斜率端点条件（zero-slope end condition）成立。对此样条画图的任务留在习题30中进行。

程序清单10-10 函数splintFE进行有固定斜率端点条件的分段三阶样条插值

```
function yhat = splintFE(x,y,xhat,fp1,fpn)
% splintFE  Cubic-spline interpolation with fixed slope end conditions
%
% Synopsis:   yhat = splintFE(x,y,xhat,fp1,fpn)
%
% Input:   x,y  = vectors of discrete x and y = f(x) values
%          xhat = (scalar or vector) x values where interpolant is evaluated
%          fp1  = slope at x(1), i.e., fp1 = f'(x(1))
%          fpn  = slope at x(n), i.e., fpn = f'(x(n));
%
% Output:  yhat = (vector or scalar) value(s) of the cubic spline interpolant
%                 evaluated at xhat.  size(yhat) = size(xhat)

% --- Set up system of equations for b(i)
x = x(:);  y = y(:);  xhat = xhat(:);  %  convert to column vectors
n = length(x);
dx = diff(x);                   %  vector of x(i+1) - x(i) values
divdif = diff(y)./dx;           %  vector of divided differences, f[x(i),x(i+1)]

alpha = [0; dx(1:n-2); 0];                   %  sub diagonal
beta  = [1; 2*(dx(1:n-2)+dx(2:n-1)); 1];     %  main diagonal
gamma = [0; dx(2:n-1); 0];                   %  super diagonal
A     = tridiags(n,beta,alpha,gamma);        %  Sparse, tridiagonal matrix
delta = [ fp1; ...
          3*(divdif(2:n-1).*dx(1:n-2) + divdif(1:n-2).*dx(2:n-1)); ...
          fpn ];

% --- Solve the system for b
mmdflag = spparms('autommd');       %  Store minimum degree ordering flag
spparms('autommd',0);               %  Set that flag to zero
b = A\delta;                        %  Solve the system
spparms('autommd',mmdflag);         %  Reset the minimum degree ordering flag

% --- Compute coefficients of spline interpolants
a = y(1:n-1);
c = (3*divdif - 2*b(1:n-1) - b(2:n))./dx;
d = (b(1:n-1) - 2*divdif + b(2:n))./dx.^2;
b(n) = [];                                   %  discard b(n)

% --- Locate each xhat value in the x vector
i = zeros(size(xhat));  %  i is index into x such that x(i) <= xhat <= x(i+1)
for m=1:length(xhat)    %  For vector xhat: x( i(m) ) <= xhat(m) <= x( i(m)+1 )
  i(m) = binSearch(x,xhat(m));
end

% --- Nested, vectorized evaluation of the piecewise polynomials
xx = xhat - x(i);
yhat = a(i) + xx.*(b(i) + xx.*(c(i) + xx.*d(i)) );
```

573 ~ 574

固定斜率端点条件样条的MATLAB实现 程序清单10-10中的splintFE函数使用有固定斜率端点条件的分段三阶样条对一个数据集进行插值，此函数的调用方法为：

```
yhat = splintFE(x,y,xhat,fp1,fpn)
```

其中x和y定义了将要插值的表格数据，输入参数`xhat`是在该处要计算插值式的x值的标量或向量。输出向量（或标量）`yhat`与输入向量（或标量）`xhat`长度相同。函数`splintFE`中的代码由以下五个主要部分组成：

1. 建立样条的三对角方程组（10-46）
2. 解此方程组得出斜率向量b
3. 由计算出的b使用方程（10-47）到方程（10-49）计算剩余的样条片段系数
4. 找出包含$\hat{x}$的样条片段，$\hat{x}$是要计算样条的x值
5. 计算$\hat{x}$处的样条插值式

下面依次讲解这些步骤。

建立三对角方程组 方程（10-46）中的三对角系数矩阵是稀疏的——对较大的n值，矩阵中大部分的元素是零。使用MATLAB稀疏矩阵存储格式可以节省存储空间和执行时间（参见附录B）。在`splintFE`函数中，使用NMM工具箱中的`tridiags`函数产生三对角矩阵。函数`tridiags`的输入是矩阵的大小和分别定义主对角、上对角和下对角的三个向量，这些对角向量分别是β，α和γ，它们在方程（10-41）到方程（10-43）中有定义。

解方程组 稀疏矩阵定义后，它就可以像MATLAB的其他矩阵一样使用。特别地，有稀疏系数矩阵的方程组可以用标准的反斜杠操作符来求解。方程组（10-46）的解可由下列语句得到：

```
b = A\delta;
```

其中`A`是系数矩阵，`delta`是右边的向量，`b`是斜率值的向量。在`splintFE`中，函数`spparms`的调用总是在语句`b = A\delta`周围，此函数用来控制方程组稀疏解的细节。函数`spparms`有三次调用：第一次是存储`autommd`标志的当前值；然后是将`autommd`标志设为“关闭”，以使方程不用再重新排序；最后是将`autommd`标志重设为以前的值。既然样条方程组的矩阵A为三对角矩阵，对函数重新排列就没有意义了。通过使MATLAB避免对方程进行重新排列操作，可以提高此方程组解的总体效率。

求剩余样条系数 已知向量b，用方程（10-47）到方程（10-49）可以很简单地计算样条系数向量a，c和d。惟一复杂的是b有n个元素（n是节点数），而样条中只有$n-1$个三阶片段。由于c_{n-1}和d_{n-1}的公式要用到b_n，所以必须保留b_n直到c和d计算出来。方法是先计算c和d，然后使用如下语句将向量b的长度缩短为$n-1$：

575

```
b(n) = [];
```

结果a，b，c和d的长度都为$n-1$，样条的计算因此更容易向量化。

找出合适的样条片段并求插值式 为求出$\hat{x}$处的插值式，必须确定包含$\hat{x}$的三阶片段。此工作和所需代码与程序清单10-8中的`hermint`函数相同。参考一下10.3.4节的讨论。

样条插值式求值 方程（10-38）是求样条插值式的第i个片段的公式。这与三阶Hermite插值式所用的公式[⊖]相同。参考一下关于`hermint`函数的讨论。

函数`splintFE`由NMM工具箱中的`demoSplintFE`函数来执行。函数`demoSplintFE`（此处未列出）用分界点处不同的样条斜率值产生插值式来逼近方程$y = x\exp(-x)$。

“自然”端点条件 “自然端点条件”隐含着存在这样的端点条件，它们在本质上更符合样条真正或理想的状态。也可以这样假设，这些边界条件比其他的好。实际上，自然端点条

⊖ 一般来说，三阶Hermite插值式的系数a_i、b_i、c_i和d_i有不同的值。

件比下面的非节点端点条件的精度差。这里的“自然”指的只是类似于绘图员的曲线尺（spline），即一个穿过固定参考点集来画平滑曲线的柔性棒。考虑图10-18所示的由三个销钉限制的绘图员的曲线尺的形状，超过最后约束点的细条的任何部分都没有曲率（直的）。如果细条的形状如$y = f(x)$所描述的那样，那么在最后的约束点和自由端之间的细条部分有$y'' = 0$。

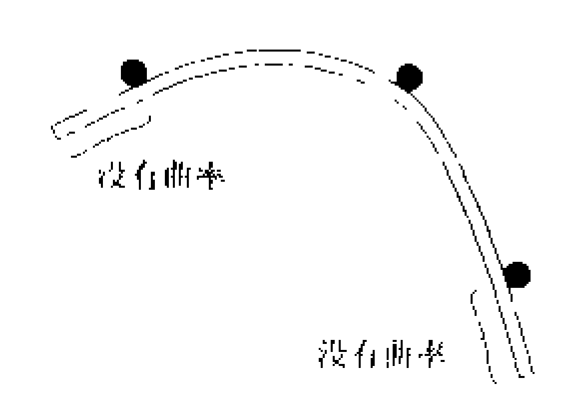


图10-18　一个由三个销钉约束的绘图员的曲线尺　柔性棒在最后的销钉和自由端之间部分的曲率为零，这就是端点条件$y'' = 0$被叫做“自然”样条的原因

通过与实际中绘图员所用的曲线尺的对比，可知三阶插值样条的自然端点条件为$y'' = 0$。由方程（10-38）中$P_i(x)$的定义，有

$$P_i''(\hat{x}) = 2c_i + 6d_i(\hat{x} - x_i) \tag{10-50}$$

576

于是在$x = x_1$处自然端点条件要求$c_1 = 0$。由方程（10-47）到方程（10-49），$c_1 = 0$反过来要求

$$\frac{3f[x_1,x_2] - 2b_1 - b_2}{(x_2 - x_1)} = 0$$

或

$$b_1 + \frac{1}{2}b_2 = \frac{3}{2}f[x_1,x_2]$$

将此条件加到方程（10-46）中，需要设

$$\beta_1 = 1,\quad \gamma_1 = \frac{1}{2},\quad \delta_1 = \frac{3}{2}f[x_1,x_2] = \frac{3}{2}\frac{y_2 - y_1}{x_2 - x_1} \tag{10-51}$$

在$x = x_n$处将方程（10-50）应用自然端点条件，需有

$$c_{n-1} = -3d_{n-1}(x_n - x_{n-1})$$

适当地代入方程（10-47）到方程（10-49）并进行简化，得

$$b_{n-1} + 2b_n = 3f[x_n,x_{n-1}]$$

方程（10-46）中的矩阵系数变成

$$\alpha_n = 1,\quad \beta_n = 2,\quad \delta_n = 3f[x_n,x_{n-1}] = 3\frac{y_n - y_{n-1}}{x_n - x_{n-1}} \tag{10-52}$$

为得到自然样条插值式，使用方程（10-41）到方程（10-44）来定义方程（10-46）中矩阵的2到$n-1$行，使用方程（10-51）和（10-52）来定义方程组中第一行和最末行。

“非节点”端点条件　当样条端点的斜率信息未知时，“非节点”端点条件是最精确的端点条件。“非节点”条件通过要求第一个内部节点（即x_2或x_{n-1}）的$P'''(x)$连续来求得。既然样条片段是三阶多项式，且样条已经要求$P(x)$、$P'(x)$和$P''(x)$在节点处连续，增加$P'''(x)$的连续性之后，说明两个邻接样条片段变成了相同的三阶多项式。此时这个内部节点就不再是两个不同三阶多项式的分界点。因此，它就不再是一个真正的节点，这就是名称“非节点”端点条件的由来。

577

首先，考虑$x=x_1$处的端点条件。由方程（10-38），

$$P_i'''(x)=6d_i$$

于是由$P_1'''(x_2)=P_2'''(x_2)$要求$d_1=d_2$。由方程（10-49）中关于d_i的定义，有

$$\frac{b_1-2f[x_1,x_2]+b_2}{\Delta x_1^2}=\frac{b_2-2f[x_2,x_3]+b_3}{\Delta x_2^2}$$

其中$\Delta x_1=x_2-x_1$，$\Delta x_2=x_3-x_2$。整理此方程使左边只有b_1项，得

$$b_1+\left(1+\frac{\Delta x_1^2}{\Delta x_2^2}\right)b_2-\frac{\Delta x_1^2}{\Delta x_2^2}b_3=2f[x_1,x_2]-2\frac{\Delta x_1^2}{\Delta x_2^2}f[x_2,x_3] \tag{10-53}$$

此方程有三个未知数b_1、b_2和b_3。在$x=x_1$处非节点条件的直接应用就是用方程（10-53）代替定义方程组（10-46）第一行的方程。但是，这种替代却破坏了系数矩阵的三对角性质。为保持系数矩阵的三对角结构[⊖]，需要从方程（10-53）中消去b_3。观察方程组（10-46）可知矩阵的第二行组成了另一个关于b_1、b_2和b_3的方程。所以，第二行的方程可用来从方程（10-53）中消去b_3。令方程（10-40）中的$i=2$可给出方程组（10-46）中第二行方程的简便形式：

$$\Delta x_2 b_1+2(\Delta x_1+\Delta x_2)b_2+\Delta x_1 b_3=3(f[x_2,x_3]\Delta x_1+f[x_1,x_2]\Delta x_2)$$

解此方程得出b_3，并将结果代入方程（10-53）中，稍作变换，得

578

$$\Delta x_2 b_1+(\Delta x_1+\Delta x_2)b_2=\frac{\Delta x_2(2\Delta x_2+3\Delta x_1)f[x_1,x_2]+\Delta x_1^2 f[x_2,x_3]}{\Delta x_1+\Delta x_2} \tag{10-54}$$

在$x=x_1$处应用非节点条件，得到方程组（10-46）中第一行的系数为

$$\beta_1=\Delta x_2 \quad \gamma_1=\Delta x_1+\Delta x_2$$

$$\delta_1=\frac{\Delta x_2(2\Delta x_2+3\Delta x_1)f[x_1,x_2]+\Delta x_1^2 f[x_2,x_3]}{\Delta x_1+\Delta x_2}$$

在$x=x_n$处应用非节点条件的过程与$x=x_1$处相同。$P'''(x)$在$x=x_{n-1}$处连续，需有

$$\frac{b_{n-2}-2f[x_{n-2},x_{n-1}]+b_{n-1}}{\Delta x_{n-2}^2}=\frac{b_{n-1}-2f[x_{n-1},x_n]+b_n}{\Delta x_{n-1}^2}$$

$i=n-1$处的方程（10-40）为

$$\begin{aligned}\Delta x_{n-1}b_{n-2}&+2(\Delta x_{n-1}+\Delta x_{n-2})b_{n-1}+\Delta x_{n-2}b_n\\&=3(f[x_{n-1},x_n]\Delta x_{n-2}+f[x_{n-2},x_{n-1}]\Delta x_{n-1})\end{aligned}$$

合并前面的两个方程消去b_{n-2}，得

$$\begin{aligned}(\Delta x_{n-2}&+\Delta x_{n-1})b_{n-1}+\Delta x_{n-2}b_n\\&=\frac{\Delta x_{n-2}(2\Delta x_{n-2}+3\Delta x_{n-1})f[x_{n-1},x_n]+\Delta x_{n-1}^2 f[x_{n-2},x_{n-1}]}{\Delta x_{n-2}+\Delta x_{n-1}}\end{aligned}$$

因此，通过将以下系数值赋给方程（10-46）中的末行，可得到$x=x_n$处的非节点端点条件：

⊖ 涉及三对角矩阵的方程组解起来要比一般矩阵方程组容易得多。

$$\alpha_n = \Delta x_{n-2} + \Delta x_{n-1}$$

$$\beta_n = \Delta x_{n-1}$$

$$\delta_n = \frac{\Delta x_{n-2}(2\Delta x_{n-2} + 3\Delta x_{n-1}) f[x_{n-1}, x_n] + \Delta x_{n-1}^2 f[x_{n-2}, x_{n-1}]}{\Delta x_{n-2} + \Delta x_{n-1}}$$

三阶样条插值的一般实现　程序清单10-11摘录了函数splint的一部分，此函数使用前面讨论的三种端点条件来构造三阶样条，它的调用可以通过下面三种方法：

```
[yhat] = splint(x,y,xhat)
[yhat] = splint(x,y,xhat,endType)
[yhat] = splint(x,y,xhat,fp1,fpn)
```

其中x和y是需要被插值的表格数据，xhat是在该处计算插值的x值的标量或向量，在xhat处插值式的值是yhat。输出向量（或标量）yhat与输入向量（或标量）xhat的长度相等。函数的第一种形式使用非节点端点条件，其中只指定了x、y和xhat；可选的endType参数是字符串natural或notKnot，分别用来选择自然端点条件和非节点端点条件；如果使用可选的fp1和fpn，那么这些值必须是x_1和x_n处的固定斜率。换言之splint(x,y,xhat,fp1,fpn)与splintFE(x,y,xhat,fp1,fpn)等价。

579

函数splint的代码篇幅比本书的一页还多。正如程序清单10-11中的注释所述，这些代码的大部分与splintFE相同。建议读者研究源代码，它包含在NMM工具箱中。此函数与splintFE的明显不同包含在程序清单10-11中。这些改变考虑了不同端点条件下对方程（10-46）系数矩阵和右边向量的更改。

例10.15　不同端点条件下样条的比较

程序清单10-12中的函数compSplinePlot使用不同的端点条件在$0 \le x \le 5$上对$y = xe^{-x}$进行三阶样条逼近。每个样条插值式的绝对误差由$E = \|\hat{y} - y_e\|_2$计算，其中$\hat{y}$是在100个点的均匀间隔上计算的样条插值式，y_e是在这些点求出的原始函数。计算插值式的100个点是任意取的。随着分界点数目的增加，插值式的值要保持不变，以作为比较各端点条件的合理依据。

程序清单10-11　函数splint部分摘录，此函数可构造出不同端点条件的三阶样条

```
function [yhat,aa,bb,cc,dd] = splint(x,y,xhat,opt1,opt2)
% splint  Cubic-spline interpolation with various end conditions
%
% Synopsis:  yhat = splint(x,y,xhat)
%            yhat = splint(x,y,xhat,endType)
%            yhat = splint(x,y,xhat,fp1,fpn)
%            [yhat,a,b,c,d] = splint(x,y,xhat)
%            [yhat,a,b,c,d] = splint(x,y,xhat,endType)
%            [yhat,a,b,c,d] = splint(x,y,xhat,fp1,fpn)
%
% Input:  x,y  = vectors of discrete x and y = f(x) values
%         xhat = (scalar or vector) x value(s) where interpolant is evaluated
%         endType = (string, optional) either 'natural' or 'notaKnot';  used
%             to select either type of end conditions.  End conditions must be
%             same on both ends. Default. endType='notaKnot'.  For fixed slope
%             end conditions, values of f'(x) are specified, not endType
%         fp1 = (optional) slope at x(1), i,e., fp1 = f'(x(1))
%         fpn = (optional) slope at x(n), i,e., fpn = f'(x(n));
```

```
% Output:  yhat = (vector or scalar) value(s) of the cubic spline interpolant
%                 evaluated at xhat.  size(yhat) = size(xhat)
%          a,b,c,d = (optional) coefficients of cubic spline interpolants

% --- Process optional input arguments
  ... see complete source code of splint.m for details
% --- Set up system of equations for b(i)
  ... same as code in splintFE

% --- Modify system of equations as appropriate for the end conditions
if strncmp('not',lower(endType),3)  %  not a knot
  A(1,1) = dx(2);     A(1,2) = dx(1) + dx(2);               % Equation for b(1)
  delta(1) = ( dx(2)*(2*dx(2)+3*dx(1))*divdif(1)...
               + dx(1)*dx(1)*divdif(2) ) /(dx(1)+dx(2));
  A(n,n-1) = dx(n-2) + dx(n-1);     A(n,n) = dx(n-2);    % Equation for b(n)
  delta(n) = ( dx(n-2)*(2*dx(n-2)+3*dx(n-1))*divdif(n-1) ...
               + dx(n-1)*dx(n-1)*divdif(n-2) ) / (dx(n-2)+dx(n-1));
elseif strncmp('nat',lower(endType),3)     %  natural end conditions
  A(1,2) = 0.5;  delta(1) = 1.5*divdif(1);                %  y''(x(1)) = 0
  A(n,n-1) = 1;  A(n,n) = 2;  delta(n) = 3*divdif(n-1);  %  y''(x(n)) = 0
elseif strncmp('fix',lower(endType),3)     %  fixed-slope end conditions
  delta(1) = yp1;  delta(n) = ypn;
else
  error(sprintf('Logic error:  endType = %s',endType));
end

% --- Solve the system for b
  ... remainder of splint is same as code in splintFE
```

580

图10-19描述了compSplinePlot函数在默认的六个点情况下的输出。对这些小数目的节点来说非节点端点条件的优势很明显；使用自然端点条件时加上零曲率的限制条件会在$x = 0$附近产生更大的误差，因为此处函数斜率变化得更快；零斜率端点条件明显地很差，但是严格斜率端点条件（exact slope end condition）使插值式和原始函数的一致性最好。

随着临界点数目的增加，运行compSplinePlot函数可得到下表所示的绝对误差。

绝对误差：$E = \|\hat{y} - y_e\|_2$

节点	自然	零斜率	非节点	严格斜率
4	7.0×10^{-2}	1.0	5.2×10^{-1}	1.6×10^{-1}
8	1.2×10^{-1}	3.3×10^{-1}	3.2×10^{-2}	5.4×10^{-3}
16	2.0×10^{-2}	1.1×10^{-1}	1.7×10^{-3}	2.4×10^{-4}
32	3.3×10^{-3}	3.5×10^{-2}	8.2×10^{-5}	1.3×10^{-5}
64	4.8×10^{-4}	1.0×10^{-2}	3.2×10^{-6}	7.4×10^{-7}
128	4.0×10^{-5}	1.7×10^{-3}	9.3×10^{-8}	4.4×10^{-8}

对固定数目的节点，如果区间端点的斜率未知，非节点端点条件产生的绝对误差最小。这与de Boor[14, pp.54-56]中的介绍一致，该书介绍了逼近误差分析。因为严格斜率端点条件在x_1和x_n处没有引入逼近误差，所以它产生的结果最好。相对于明显错误的零斜率端点条件，

581

自然端点条件的精度要高，但明显地比非节点端点条件的精度差。

程序清单10-12 函数compSplinePlot用来计算由固定斜率、自然和非节点端点条件对方程 $y = xe^{-x}$ 的三阶样条插值的精度问题

```
function compSplinePlot(n)
% compSplinePlot  Compare end conditions for cubic-spline interpolants
%                 Approximations to y = x*exp(-x) are constructed and plotted
%
% Synopsis:  compSplinePlot
%            compSplinePlot(n)
%
% Input:     n = (optional) number of knots in the range 0 <= x <= 5
%                Default:  n=6
%
% Output:    Plot of spline approximations to y = x*exp(-x) with not-a-knot,
%            natural, zero-slope, and exact-slope end conditions.  Normalized
%            errors for each interpolant are also computed and printed

if nargin<1, n=6;  end

x = linspace(0,5,n)';                 %  Generate discrete data set
y = x.*exp(-x);
xi = linspace(min(x),max(x))';        %  Evaluate spline at these xi
ye = xi.*exp(-xi);                    %  Exact f(x) at the xi

yi = splint(x,y,xi,'natural');        %  Spline with natural end conditions
errNat = norm(yi-ye)
subplot(2,2,1);  plot(x,y,'bo',xi,ye,'b-',xi,yi,'r--');  axis([0 6 0 0.5]);
legend('knots','spline','x*exp(-x)');  title('Natural end conditions');

yi = splint(x,y,xi,0,0);              %  Spline with zero-slope end conditions
errz = norm(yi-ye)
subplot(2,2,2);  plot(x,y,'bo',xi,ye,'b-',xi,yi,'r--');  axis([0 6 0 0.5]);
legend('knots','spline','x*exp(-x)');  title('Zero-slope end conditions');

yi = splint(x,y,xi);                  %  Spline with not-a-knot end conditions
errNot = norm(yi-ye)
subplot(2,2,3);  plot(x,y,'bo',xi,ye,'b-',xi,yi,'r--');  axis([0 6 0 0.5]);
legend('knots','spline','x*exp(-x)');  title('Not-a-knot end conditions');

yp1 = (1-x(1))*exp(-x(1));            %  Exact slope at x(1)
ypn = (1-x(n))*exp(-x(n));            %  and at x(n)
yi = splint(x,y,xi,yp1,ypn);          %  Spline with exact-slope end conditions
errExs = norm(yi-ye)
subplot(2,2,4);  plot(x,y,'bo',xi,ye,'b-',xi,yi,'r--');  axis([0 6 0 0.5]);
legend('knots','spline','x*exp(-x)');  title('Exact-slope end conditions');
```

582

10.4 MATLAB的内置插值函数

表10-2列出了标准MATLAB工具箱中的插值函数。除interpft外，所有这些函数都是进行分段插值的。这里没有与函数lagrint和newtint等价的内置函数，这两个函数使用任意阶数的多项式进行一维插值。 583

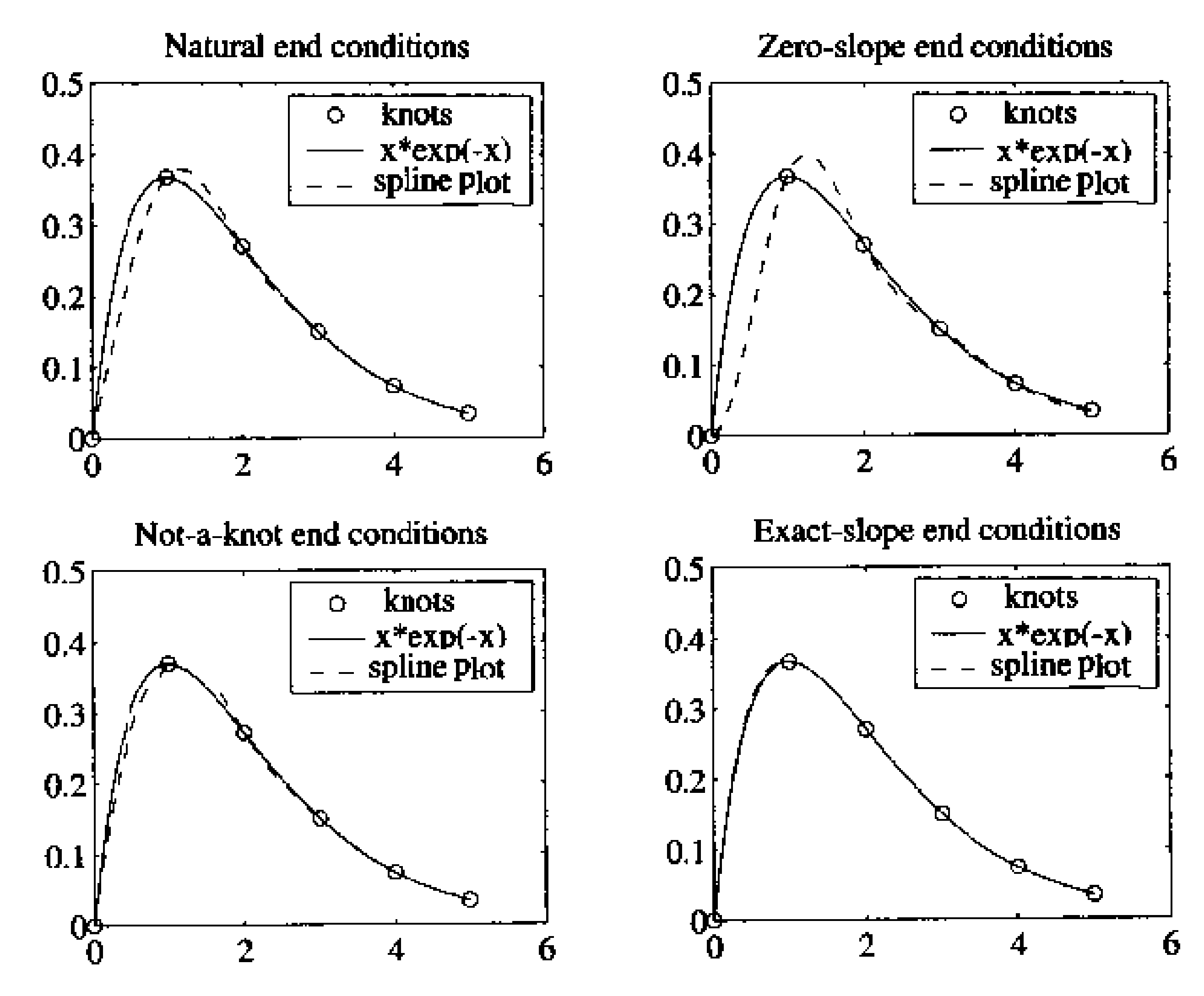


图10-19　使用有不同端点条件的三阶样条对$y = xe^{-x}$进行的逼近，本图是由程序清单10-12中的函数compSplinePlot完成的

表10-2　MATLAB中进行数据插值的内置函数

函　数	描　述
interp1	使用分段多项式进行一维插值
interp2	使用最临近、双线性、双立方插值式进行二维插值
interp3	使用最临近、双线性、双立方插值式进行三维插值
interpft	使用傅立叶级数（FFT）对等距分布数据进行插值
interpn	对interp3进行n维扩展
spline	用非节点或固定斜率端点条件的三阶样条进行一维插值

使用interp1和spline进行一维插值

内置interp1函数可使用下面四种方法中的一种进行一维插值:

- 最临近（*nearest-neighbor*）插值使用常数分段函数（零阶多项式）。插值式在两邻接节点的中点处不连续。
- 线性插值使用分段线性多项式。
- 三阶插值使用分段三阶多项式并要求插值式及其导数连续。
- 样条插值使用带非节点边界条件和固定斜率边界条件的三阶样条。这与内置的spline函数的插值相同。

图10-20给出了这些插值方法的一个样例。函数interp1可由以下方法调用:

```
yhat = interp1(y,xhat)
yhat = interp1(x,y,xhat)
yhat = interp1(x,y,xhat,method)
```

在函数interp1的每种调用形式中，y是要插值的表格函数，*xhat*是在该处要计算插值

的x值的标量或向量。如果向量x已知，它就会为表格函数提供自变量的值。也就是说，x是$y=f(x)$表中的x值的向量。如果x不是输入参数，就可以假设表格有`x =1:n`定义的x值，其中n = `length(y)`。可选的第四个参数*method*为字符串值"nearest"，'linear"，"cubic"或"spline"中的一个，这些字符串分别对应于前面列表中描述的四个方法。输出*yhat*是一个与*xhat*的长度相等的向量（或标量），它包含xhat每个元素的插值式的值。 584

下面的语句示范了怎样使用interp1进行最临近插值：

```
>> x = linspace(0,1.5,10);              % define knots
>> y = humps(x);                        % humps is a built in function
>> xi = linspace(min(x),max(x));        % eval interpolant at xi
>> yn = interp1(x,y,xi,'nearest');
>> plot(x,y,'o',xi,yn,'-');
```

结果如图10-20的左上图。图10-20中的四个图由NMM工具箱的demoInterp1函数得到。

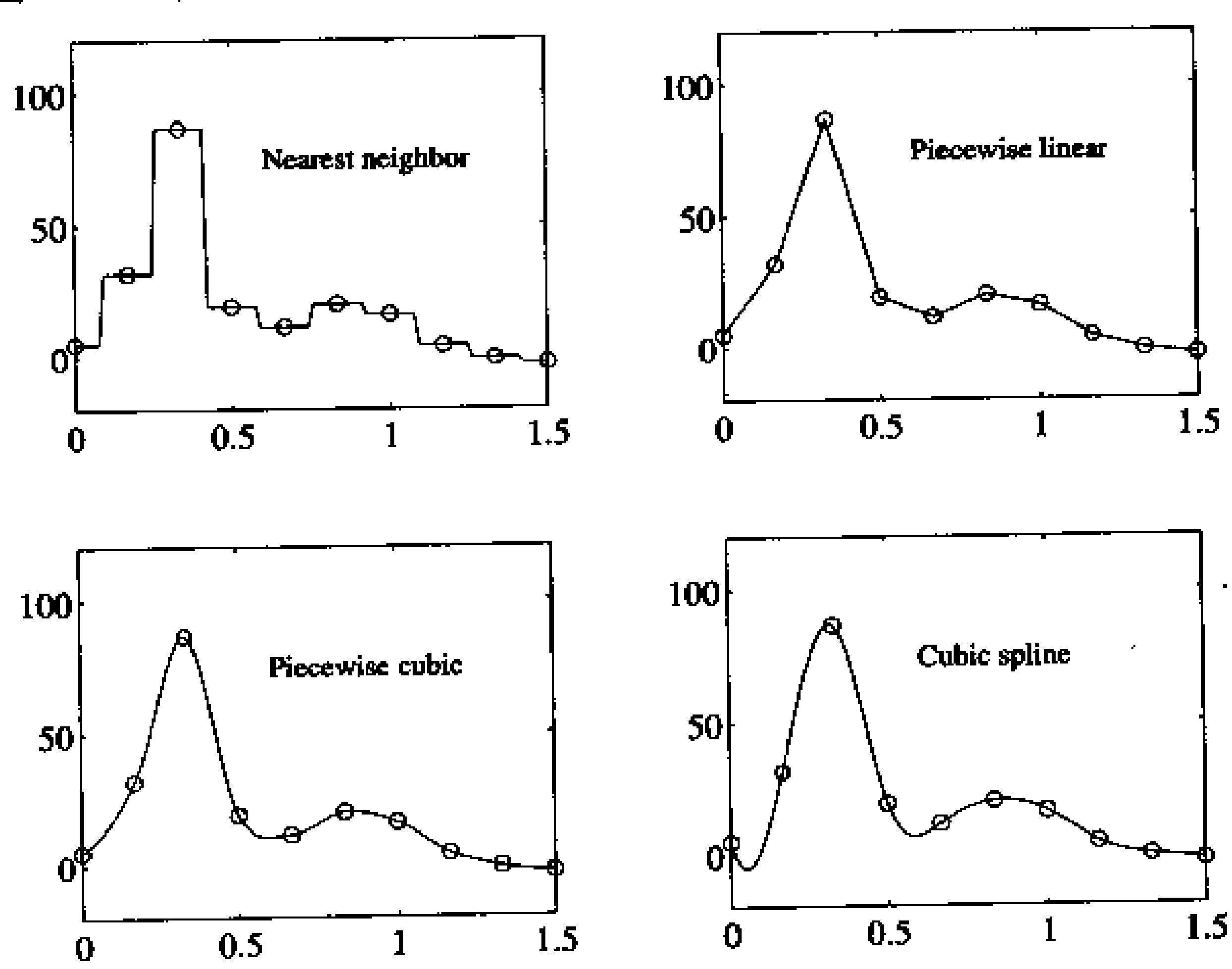


图10-20 使用内置interp1函数提供的四种方法对相同数据集进行插值，本图结果由NMM工具箱的demoInterp1函数完成。

585

三阶样条插值可以使用interp1或spline来进行。语句

```
yhat = interp1(x,y,xhat,'spline')
```

等价于语句

```
yhat = spline(x,y,xhat)
```

输入参数x和y给出了定义插值函数的数据表。插值式*yhat*的值在每个*xhat*处都要计算。缺省情况下，interp1(...,'spline')和spline(...)使用非节点端点条件。如果输入向量y比输入x多两个元素，那么y的第一个和最后一个元素的值就分别为$y'(x_1)$和$y'(x_n)$的值。

注意在图10-20的右下角三阶样条插值式在x = 0附近有一处下凹。这并不能反映humps函数的局部趋势。误差是由节点的分布不均匀和非节点端点条件引起的，可以通过在x = 0附近添加节点或使用严格斜率端点条件来改进（参见练习33）。

10.5 小结

本章讨论了两个主题：任意阶的多项式插值和分段多项式插值。分段多项式插值一般在实际应用中更有用，因为它在数值求解上难度较小。

表10-3列出了本章开发的m文件插值函数。标准MATLAB工具箱提供的插值函数已在表10-2中列出。

任意阶的插值多项式

构造任意阶的插值多项式时，需要选择基本函数。本章中我们考虑了单项式、拉格朗日和牛顿基本插值公式。

求单项式基本插值公式的系数需要解Vandermonde方程组。Vandermonde矩阵可能是病态的，这样会导致单项式系数不确定。另外，单项式中的各项（系数乘以x的对应次幂）可能在大小上有很大差异，这就导致了多项式计算中的舍入误差。这些数值上的复杂性可在构造[586] Vandermonde方程组之前通过对数据x进行转换和缩放来改进（见例10.5和练习4）。

表10-3　本章开发的用于插值的函数。页码中带N.A.（不可用）的没有在书中列出，但包含在NMM工具箱的interpolate目录下

函　数	小　节	描　述
binSearch	10.3.2	使用折半搜索算法来查找满足$x_i \le \hat{x} \le x_{i+1}$的下标$i$
compInterp	N.A.	比较采用不同的基本公式多项式插值的浮点数(参见例10.10)
compSplinePlot	10.3.5	比较使用不同端点条件的三阶样条在逼近$y=xe^{-x}$时的精度
demoGasLag	10.2.2	使用拉格朗日多项式基本插值来对汽油价格数据进行插值
demoGasNewt	N.A.	使用牛顿多项式基本插值来对汽油价格数据进行插值
demoGasVand	N.A.	使用单项式基本插值来对汽油价格数据进行插值，这需要求解一个很难成比例的Vandermonde方程组
demoGasVandShift	N.A.	使用单项式基本插值来对汽油价格数据进行插值，其中对数据进行了变换，Vandermonde方程组可以更好地缩放并有更小的条件数
demoHermite	10.3.4	函数$y=xe^{-x}$在$0 \le x \le 5$内的分段三阶Hermite插值
demoInterp1	N.A.	使用内置的interp1函数来对humps函数产生的数据样本进行插值
demoWiggle	N.A.	画出由于多项式插值式的阶数增加而引起的摆动图（参见例10.11）
divdiffTable	10.2.3	建立一个均差系数表
hermint	10.3.4	进行分段三阶Hermite插值
lagrint	10.2.2	使用任意阶数的拉格朗日多项式进行插值
linterp	10.3.4	对(x, y)数据表进行分段线性插值
newtint	10.2.3	使用任意阶数的牛顿多项式进行插值
splint	10.3.5	使用不同的端点条件进行三阶样条插值
splintFE	10.3.5	使用固定斜率端点条件进行三阶样条插值

拉格朗日基本插值公式插值在理论分析中最有用。由拉格朗日基本插值公式形成的多项式插值式的舍入误差比较小，但是计算公式比较繁琐。

对数值计算来说，使用牛顿基本插值公式进行插值比使用单项式或拉格朗日基本插值公式更好。牛顿多项式不但舍入误差比较小，而且计算公式也很高效。推荐使用程序清单10-4中的newtint函数来进行任意阶多项式的插值。牛顿插值多项式的系数是输入数据集的均差。均差在推导分段Hermite多项式和三阶样条时也很有用处。

对均匀分布的支点数据进行任意阶的多项式插值时，必须防止由多项式摆动带来的误差。多项式摆动会影响任何基本插值公式上定义的多项式。

587

分段多项式插值

分段多项式插值使用相对低阶的多项式，它们定义在输入数据的子集上。对一维数据（$y = f(x)$）来说，插值式是由很多插值式在某些点上连接而成的，这些沿着x轴的点称为分界点或节点。若计算点$\hat{x}$处的分段插值式，需要额外的代码逻辑来寻找定义在所有节点上的分段插值式集合中合适的局部插值式。

分段线性插值可以由程序清单10-7中内置的interp1函数或linterp函数实现。函数interp1也提供了最邻近、分段三阶和三阶样条插值式。

分段三阶Hermite插值需要指定每个节点的函数$y=f(x)$及其一阶导数$f'(x)$，这对有解析表达式的函数来说是非常可行的。当表格数据构成的函数的导数不存在时，推荐使用三阶样条插值。三阶样条由于插值式及其一阶导数在节点处连续，所以是平滑的。虽然三阶样条包含了局部定义的三阶片段，但是节点处二阶导数连续，这样就可以将邻接片段连接起来，因此，构造三阶样条插值式需要解方程组。另外，还需提供样条末端的斜率条件。内置的interp1或spline函数只考虑使用非节点和固定斜率端点条件构造三阶样条。程序清单10-11中的splint函数可用非节点、固定斜率或自然端点条件构造三阶样条。

建议

选择插值策略的第一步就是了解应用的需要。你需要在表格中查找值么？或者你需要构造反复计算逼近值的程序么？这两种情况下，分段插值都可能是一个很好的选择。在条件有限的情况下，构造固定阶数的插值多项式可能会是一种更简洁（即自包含）的解决方案。对后面的情况，推荐使用牛顿多项式插值形式。可以先用divDiffTable构造插值式系数，然后硬编码（hard-code）进一个小m文件来计算插值式。在做这些工作时，应该确定由divDiffTable返回系数的所有有效位数。

对表格数据的常规插值，推荐使用内置的interp1和interp2函数[⊖]。内置程序也支持外插。

588

如果插值式的总体平滑很重要，就应该考虑三阶样条或分段三阶Hermite插值。样条插值式的精确度受端点条件的影响很大。若缺乏数据端点处的斜率信息，应该使用非节点端点条件。如果端点处的斜率已知，就使用固定斜率形式，其实现函数有NMM工具箱中splint函数、内置的interp1(...,'spline')函数和等价的spline函数。用户若需要更复杂的插值工具，可以考虑Mathworks公司出售的样条工具箱（*Spline Toolbox*）。

对二维或更高维数的插值，应该使用interp2和interp3函数。这些函数要求数据定义在矩形网格中，网格无须一致。可使用griddata函数对$z = f(x, y)$定义的分散数据进行插值，其中x和y坐标不在矩形网格上。

补充读物

插值是数值分析领域的一个主要部分。本章的介绍与Conte和de Boor[11]中的介绍非常相

⊖ 对于三维及三维以上的插值，将用到interp3和interpn函数，但这就不是常规插值了。为了避免盲目地使用这些高维数的插值函数，你应该提醒自己：“有没有一种和数据联系密切的插值方法？”

近，本章的一些材料与Van Loan[77]也比较类似，而Cheney and Kincaid[10]对这些材料作了更详尽的介绍，只是其中的样条推导是基于系数的二阶导数求解的，而不是在节点处进行一阶求导。严格数学意义上的插值计算，可以参见Boor[14]和Stoer and Bulirsch[70]。插值在计算机辅助设计（CAD）中的应用，可参考Farin[21]。曲线拟合中样条插值的应用，可参见de Boor[14]和Dierckx[17]。

本章中没有涉及到的两个重要主题是插值式计算中的*B*样条和*Neville*算法。*B*样条函数是构成所有样条的数学基础，关于它的介绍可参考[10、14、21、43]。Neville算法可以不用明确构造多项式的系数而计算任意阶的多项式插值式，当只需要一个插值式的值时，使用它进行插值和外插非常高效。另外，Neville算法也可以用在Romberg积分中，这是本章没有涉及到的另一主要主题（更多信息见[10、61、64、70]）。

习题

589

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1. (1－)找出线性插值多项式$P_1(x) = c_1x+c_2$的基本函数。
2. (2)Eckert and Drake（*Analysis of Heat and Mass Transfer*,1972,McGraw-Hill,Table B-13,p.790）给出了下列水的表面张力对温度的函数数据：

T(°C)	0	10	20	30	40	50	60	70	80	100
$\sigma\times10^3$(N/m)	78.2	74.8	73.4	71.6	70.2	68.5	66.7	64.9	63.2	59.3

例如，σ在0°C的值是78.2×10^{-3}N/m。画出σ相对于T的关系图。这些数据适合插值吗？对这些数据的逼近还有什么其他更合适的方法？

3. *(1+)例10.4和例10.5中Vandermonde矩阵的条件数是多少？$\kappa(A)$的改变与例10.5中得到的不同结果有什么关系？
4. (2)使用规格化年份来代替变换年份，对例10.4和例10.5中的汽油价格数据进行单项式基本插值。年份的规格化公式为$y_n=(y-\bar{y})/(y_{max}-y_{min})$，其中$\bar{y}$是$y$的平均值。求$y_n$值的范围。将插值系数的大小和结果Vandermonde矩阵的条件数与例10.4和例10.5中的比较。
5. (2－)在例10.4中，将油价由美分转化成美元会改进方程组的条件数吗？解释具体原因。
6. *(2+)在例10.4中，插值多项式的系数使用五位有效数字进行计算和打印。使用截断系数以及如下定义求出插值式中的油价和误差：令(y_i, p_i)为已知表格数据的年份和价格。令$\hat{p}_i$为y_i处使用未截断系数插值所得价格，$\bar{p}_i$为y_i处使用截断系数插值所得价格。没有数值误差时，我们希望由插值的定义有$p_i=\hat{p}_i=\bar{p}_i$。计算并打印p_i, $\hat{p}_i-p_i$, $\|\hat{p}-p\|_2$, $\bar{p}_i$, $\bar{p}_i-p_i$以及$\|\bar{p}-p\|_2$。系数c需要保留多少位有效数字得到的$\|p-\bar{p}\|_\infty$才能近似于$\|p-\hat{p}\|_\infty$（提示：参考NMM工具箱utils目录下的chop10函数）。
7. (2+)用例10.5中叙述的变换多项式插值式系数重复上个练习。
8. (1+)写出穿过(x_j,y_j)、(x_{j+1},y_{j+1})、(x_{j+2},y_{j+2})和(x_{j+3},y_{j+3})的三阶插值多项式的拉格朗日基本函数。
9. (3)在10.2.2节介绍的拉格朗日插值实现中，曾声明方程（10-15）中的分母可以进行预先计算并存储在下三角矩阵中，此矩阵元素只用两次。证明为何要使用下三角矩阵存储，以及此矩阵中的元素为何只使用两次。

590

10. (2)使用牛顿插值形式构造三阶插值多项式系数的4×4方程组（见方程（10-22））。手工解此方程组求出方程（10-23）中的系数，要求列出所有的中间步骤。

11. (1+)手工构造例10.7中的插值问题的均差表。

12. * (1+)对例10.7中的下列支点数据手工进行二阶插值:

(a) $T_1=20, T_2=30, T_3=40$

(b) $T_1=0, T_2=10, T_3=20$

将结果与$T_1=10, T_2=20, T_3=30$所得的插值结果进行比较。

13. (2)手工对例10.7中22°C下的粘度数据进行三阶和四阶多项式插值。你会使用哪些支点? 结果与本例中的二阶插值有明显不同吗?

14. * (2) NMM工具箱data目录下的H2Osat.dat文件包含了水的饱和数据。使用这些数据和二阶多项式插值，手工计算30°C≤T≤35°C下的p_{sat}。

(a) 手工构造均差表，使用divDiffTable函数来检查计算结果。

(b) 从均差表中析取出牛顿插值多项式的系数。

(c) 计算T=32°C、33°C和34°C处的插值式。用newtint函数来检查计算是否正确。

15. (2)用三阶多项式插值重复练习14。用二阶和三阶多项式插值得到的p_{sat}值有什么不同?

16. (3)建立一个定制的，自包含的程序，返回作为温度函数的甘油粘度。程序使用例10.7中提供的数据。此程序应该计算一个牛顿基本函数的三阶多项式。首先，使用divDiffTable函数来计算插值多项式的系数，将这些系数的值作为一个向量存储在你的程序中，然后计算牛顿多项式。换言之，系数由你——程序员——来计算，且只计算一次，而不是每次在程序执行时计算。这种方法的好处是粘度计算不需要NMM工具箱中的其他程序。此程序也可以转变成其他语言的实现，如C、Java或Fortran。

17. (3)考虑下面的代码段，它使用牛顿多项式插值来逼近在区间0≤x≤2π上的正弦函数。

```
x = 0:pi/6:2*pi;      y = sin(x);   % original data
xi = linspace(min(x),max(x));       % eval interpolant at xi

yi2 = newtint(x(1:3),y(1:3),xi);   % 2nd degree interpolant
yi3 = newtint(x(1:4),y(1:4),xi);   % 3rd  "  "    "  "
yi4 = newtint(x(1:5),y(1:5),xi);   % 4th
yi5 = newtint(x(1:6),y(1:6),xi);   % 5th
yi6 = newtint(x(1:7),y(1:7),xi);   % 6th
plot(x,sin(x),'o',xi,yi2,'.',xi,yi3,'--',xi,yi4,':',xi,yi5,'-',...
     xi,yi6,'-.');
legend('original','degree 2','degree 3','degree 4','degree 5',...
       'degree 6');
axis([0 2*pi -5 5]);
```

591

执行这些语句可知，所尝试的插值在较大的x值上结果很差。增加阶数不一定会增加计算的精度（四阶插值看起来比五阶插值要好）。用户在使用newtint时就已经产生了一个错误。这段代码有什么问题? 写出一段正确的代码，使它能产生整个0≤x≤2π区间上的良好的插值结果。阶数大于五时，正确的插值结果几乎都一样（提示：newtint允许使用外插方法吗? ）。

18. * (3+)函数lagrint和newtint所用多项式插值式的阶数由这些函数的输入向量长度决定。假设用户希望指定插值式的阶数而不考虑输入数据的长度，换言之，就是假设用户要在长度为m的表格中进行n阶的局部插值，其中$m>n$。这需要在输入数据表中选择一个合适的子集，然后将此子集传给lagrint或newtint函数。例如，二阶插值可以用

```
x = ...     % define tabular data
y = ...
xhat = ...     % interpolate at this value
ibeg = ...     % beginning index for support points of interpolant
yhat = newtint(x(ibeg:ibeg+2),y(ibeg:ibeg+2),xhat)
```

来执行。写一个名为quadinterp的m文件函数，它能自动选择向量x和y的合适子集，并返回这些支点上二阶插值式的值。quadinterp的定义语句是：

```
function yhat = quadinterp(x,y,xhat)
```

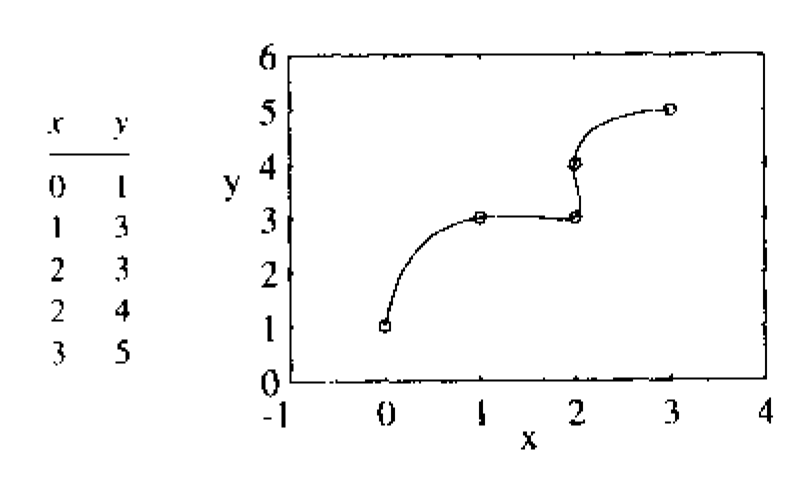
函数quadinterp调用函数newtint来进行插值。可参考程序清单10-6中的binSearch函数。使用quadinterp函数来产生例10.2中甘油粘度数据的平滑曲线。

19. (3+)使用练习18中的成果来开发一个polyinterp函数，使它能对多于n+1个数据对的表进行n阶多项式插值。函数polyinterp的定义语句为：

```
function yhat = polyinterp(x,y,xhat,n)
```

其中n是多项式插值式的阶。证明polyinterp(...,2)与quadinterp(...)完全相同。用你的polyinterp函数对例10.2中的甘油粘度数据求三阶插值式。对原数据与插值式进行图示。注意，此问题的一个精美的解被称为Neville插值。 [592]

20. (2+)考虑有重复数据的给定数据表的插值问题，其中一种插值方法如下图中曲线所示。表中的数据有问题，因为当数据由$y=f(x)$表示时，一些x的$f(x)$值并不单一。数据用$x=g(y)$来描述时，问题仍然存在。这种类型的插值很重要，例如，使用CAD程序来描绘一个物体的形状。



(a) 构造这些数据的Vandermonde方程组。系数矩阵的秩是多少？如果第四个点由(2,4)变为(2,3)，系数矩阵的秩又是多少？

(b) 在(a)中出现的问题可以用拉格朗日或牛顿基本插值公式来修正吗（直接使用lagrint和newtint进行实验会提供一些信息）？使用内置的interp1函数可以解决这个问题吗？

21. *(3)多项式摆动的经典例子是所谓的Runge函数：

$$r(x)=\frac{1}{1+25x^2} \qquad (10\text{-}55)$$

它由德国数学家Carl Runge(1856—1927)而得名，此人使用这个函数来研究高阶多项式插值的特性。写出一个名为runge的m文件函数来完成以下任务。

(a) 计算n个在区间$-1\leqslant x\leqslant 1$均匀分布的x_k值（$k=1,\cdots,n$）。令n为runge函数的一个输入参数。

(b) 由方程（10-55）计算$r(x_k)$，$k=1,\cdots,n$。

(c) 由n对$(x_k, r(x_k))$定义$n-1$阶多项式插值式P_{n-1}。求出区间$-1 \leqslant x \leqslant 1$上点$\hat{x}_j$，$j=1,\ldots,100$处的插值式的值。

(d) 画出原始数据$(x_k, r(x_k))$、插值式$(\hat{x}_j, P_{n-1}(\hat{x}_j))$和函数的真实值在插值点$(\hat{x}_j, r(\hat{x}_j))$的比较图，规定$r(x_k)$为空心圆，$P_{n-1}(\hat{x}_j)$为虚线，$r(\hat{x}_j)$为实线。

593

(e) 打印$\|r(\hat{x})-P_{n-1}(\hat{x})\|_2$的值。

在`n = 5:2:15`条件下运行你的函数，讨论随着n的增大$\|r(\hat{x})-P_{n-1}(\hat{x})\|_2$的变化。

22. (3)从上个练习中建立的`runge`函数，写一个`rungec`函数。这里不使用均匀分布的点，而是使用Chebyshev点。

$$x_i = -\cos\left(\frac{(2k-1)\pi}{2n}\right), \qquad k = 1,\cdots,n$$

当使用Chebyshev点来代替均匀分布的点时，比较$\|r(\hat{x})-P_{n-1}(\hat{x})\|_2$的性质。

23. (2)重写程序清单3-12中的`H2Odensity`函数，对NMM工具箱`data`目录下的`H2Odensity.dat`文件中的密度数据进行插值。

24. *(2) NMM工具箱的`data`目录下有一个`stdatm.dat`文件，它给出了所谓标准大气压的属性作为海拔高度的函数的数据。写一个程序，用分段线性插值返回表格中任何高度上的T、p和ρ的值。写出自己的m文件函数，使它在执行时不需要读m文件中的数据，也就是说，将数据存储在你的m文件的矩阵变量中。在z = 5500m和z = 9023m处T、p和ρ的值是多少？画出$0 \leqslant z \leqslant 15$km范围内$T$和$\rho$的变化图。

25. (2+)去掉函数`binSearch`中的输入量范围检查部分（将这些语句变成注释），令输入为`x = 0:10`和`xhat = 11`，运行修改过的`binSearch`函数。描述一下所产生的结果，并评价此结果会如何影响接下来的插值步骤。

26. (2)修改`binSearch`函数使其在`xhat < x(1)`时返回`ia = 1`，在`xhat > x(n)`时返回`ia=n -1`。修改后，`binSearch`函数可用于插值和外插中，用来返回下标。

27. (2+)在`binSearch`函数添加一个额外的检查，用来验证数据是否是单调的（提示：使用`diff`函数来计算`dx`值的向量。检查`dx`元素的符号以确定单调性）。设计至少两组单调输入数据，验证对`binSearch`函数的修改是有效的。

28. (2+)完成方程（10-37）中求系数a_i、b_i、c_i和d_i所需的代数操作。

29. (2+)通过直接计算的办法来验证三阶样条插值式中$P_{i-1}(x_i) = P_i(x_i)$和$P'_{i-1}(x_i) = P'_i(x_i)$（提示：使用方程（10-39）中的系数以及$P(x)$和$P'(x)$的定义）。

30. (2+)使用`splintFE`函数重复例10.14中的计算，画出样条图。

31. (2+)考虑在一个值上计算分段三阶Hermite插值和三阶样条插值的情况。函数`hermint`先计算所有三阶片段的方程，然后再找到插值所需要的合适片段。但是，也可以只建立合适片段的插值式，这样计算三阶Hermite插值的程序更高效。对三阶样条插值式还能同样节省效率吗？解释具体原因。

594

32. *(1+)画出下列数据的样条插值式。

x_i	0.2	0.6	1.0	1.4	1.8	2.2
y_i	0.5535	1.0173	1.0389	0.8911	0.7020	0.5257

将你的图与$y = \sqrt{12.5x}\exp(-\sqrt{1.5x})$进行比较，上表中的数据就是用它产生的。

33. (2+)图10-20的右下图中，humps函数的三阶样条插值式在$x = 0$附近有一处下凹。求出在区间$0 \leqslant x \leqslant 1.5$内100个点上的humps函数并画图，以此作为参考。在相同的坐标轴上，叠加严格斜率端点条件和非节点端点条件的三阶样条插值式的图形。用内置的spline函数产生10个均匀分布节点的样条。当样条由17个均匀分布节点产生时，重复上述比较。在你的计算和图形的基础上，推荐一个程序来产生比较精确的humps的样条插值式。

34. (2+)下面的语句构造内置humps函数在区间$0 \leqslant x \leqslant 1.5$上的三阶样条插值式。

```
x = linspace(0,1.5,10)';
y = humps(x);
xhat = linspace(min(x),max(x),36)';
yhat = spline(x,y,xhat);
plot(x,y,'o',xhat,yhat,'r-')
```

观察产生的图形可知，实线并没有经过humps紧邻的第一个顶点的节点（添加一条语句axis([0 0.5 80 90])可以更清楚地看见）。插值式之间的不一致是由于样条端点条件不同造成的吗？或者内置的spline程序有bug？或有其他不一致的原因？给出这个问题的正确解法。

35. (3-)下面的语句构造了有理数函数$y = (1 - x^2)/(2+x^3)$在区间$0 \leqslant x \leqslant 10$上的三阶样条插值式。

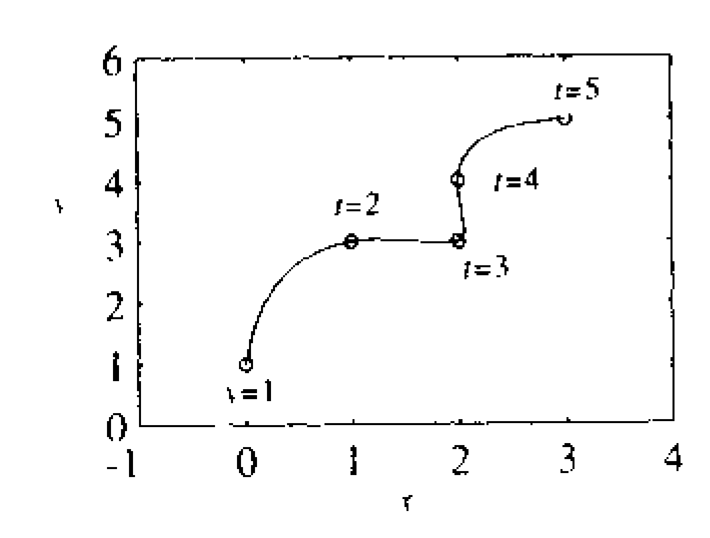
```
n = 5;
x = linspace(0,10,n);
y = (1 - x.^2)./(2 + x.^3);
xhat = linspace(min(x),max(x));
[yhat,a,b,c,d] = splint(x,y,xhat,'natural');
...
```

添加语句使用系数向量a、b、c和d来计算并打印出样条的$y''(x_1)$和$y''(x_n)$。$y''(x_1)$和$y''(x_n)$的值应该是多少？如果$y''(x_1)$和$y''(x_n)$的值并不是你所期望的，那么能断定spline函数中有bug吗？

595

36. (3+)写出splint函数的修改版（以及其他可能需要的函数），使其能在xhat<x(1)和xhat>x(n)上进行外插。用修改的函数对例10.3中的航空乘客数据进行三阶多项式外插。

37. (3)练习20中包含了有重复的x和y的数据集，此练习包含的图由参数样条（*parametric spline*）产生。参数样条不使用$y = f(x)$，而是用$x=x(t)$和$y=y(t)$进行插值，其中t是曲线的参数，如下所示：



对图中的数据，t只是用来作表中数据点的索引。t不一定是单值的和单调递增或递减的．另外，t也不一定是整数。

(a) 用下面函数的序来开发出一个paramSpline函数：

```
function [xi,yi,ti] = paramSpline(x,y,n)
% paramSpline  Parametric spline interpolation.
%
% Synopsis:    [xi,yi,ti] = paramSpline(x,y)
%              [xi,yi,ti] = paramSpline(x,y,n)
%
% Input:  x,y = vectors of data defining a curve in (x,y) plane
%         n   = (optional) number of points to generate on the
%                interpolated curve.  Default:  n = 100
%
% Output: xi,yi = vectors of points on parametric curves x = xi(ti)
%                 and y = yi(ti) obtained from separate cubic spline
%                 interpolations of x and y.  Plotting (xi,yi)
%                 produces a smooth curve through the (x,y) data
%
%         ti = vector computed with ti = linspace(min(t),max(t),n)
%              where t = 1:length(x) is vector of indices for x and
%              y, and n is an optional input parameter.
```

596

(b) 使用你开发的paramSpline函数和练习20中的数据画出练习20中的图形。

第11章 数值积分

如果涉及初等函数的积分$\int f(x)\mathrm{d}x$（不是其他积分）找不到其有由初等函数构成的解析表达式，或者只在一些离散的x点上知道函数$f(x)$的值（例如，$f(x)$是由实验数据或计算机的模拟输出得到的函数关系），那么就必须对定积分

$$I=\int_a^b f(x)\mathrm{d}x \tag{11-1}$$

的值进行数值逼近（近似）。数值积分在积分的解析式已知时也很有用，但是这样做计算起来会有困难或不方便。

本章主题：

1. **基本思想和术语**

   介绍了数值积分常用的方法和术语。

2. **Newton-Cotes公式**

   Newton-Cotes公式是在等距点上对被积函数的逼近多项式进行积分的方法。梯形公式和Simpson公式就是常用的例子，本章介绍了这些方法，并给出了m文件实现。

3. **高斯求积法**

   高斯求积法是对定义在确定点或节点（*node*）上被积函数的近似正交多项式进行积分的方法。对节点进行选择，为的是给出最高次多项式的精确值。这里推导了高斯求积法的理论，并提供了使用该方法对用户自定义被积函数进行积分的m文件。

4. **自适应求积法**

   自适应求积法(Adaptive Quadrature)会自动选择被积函数要计算的点，试图在用户自定义容差范围内来计算积分。这里推导了简单的自适应求积法，并示范了内置函数quad和quad8的使用。

5. **特殊积分和并发问题**

   这里示范了一些处理特殊积分以及被积函数有奇异点或有无穷大极限积分的方法。

图11-1 第11章的主题

对数值积分方法的选择多少与$f(x)$的性质（类型）有关。本章提供了几个实现不同积分方法的m文件函数。所有的数值积分方法都要求$f(x)$可以在区间$a\leqslant x\leqslant b$的任意x上求值。如果$f(x)$有奇异点（即对区间上的某些x^*来说有$f(x^*)=\pm\infty$），就要特别警惕。如果积分的极限中至少有一个是无穷的，就必须使用其他程序。这些特殊问题现在已经有很好的解决方法，但是还没有任何一种数值积分法可以适用于所有的积分。

597/598

图11-1给出了本章涉及的主题摘要。本章开始大概介绍了一些基本术语和不同类型的积分方法，然后在各节中对不同的积分方法进行推导。方法介绍的顺序通常是从简单到复杂的。

特殊函数积分形式简注

通常用有积分形式的特殊函数（如erf (x)和$J_n(x)$）来检验本章的积分方法。这里给出了很多表格数据表示的特殊函数（例见[1]）。如果子程序、自动计算需要这些特殊函数的值，就不要对它们直接作数值积分。积分方法的成功与否很可能决定于特殊函数的参数。要保证任何参数值上的数值积分的精度，就需要进行仔细测试。最后，你需要重复那些天才的前辈们做过的艰难工作，他们首先推导出了参照表，虽然参照表中的一些值可能已经由数值积分产生，但是仍然要用到其他的技术，如有理函数逼近。有很多处理特殊函数的库函数，实际使用中

应该首先利用它们。例如，`erf`和`bessel`函数族就是标准MATLAB工具箱中的一部分。Press 等人在文献[61]中提供了一些代码，并对如何计算特殊函数作了比较优秀的概括。`netlib`(www.netlib.org)的`specfun`目录下有特殊函数的程序。有兴趣的读者可以研究函数`beta`、`ellipj`、`erf`、`expint`和`gamma`的MATLAB源代码。

例11.1 制造椭圆管

图11-2描述了用平板材料制造椭圆管道的成形过程。为构成一个轴长分别为a和b的椭圆，首先要找一个宽为$L = 2aE$的薄板，其中E是完全椭圆积分

$$E = \int_0^{\pi/2} \sqrt{1 - k^2 \sin^2 \theta}\, \mathrm{d}\theta \tag{11-2}$$

$k^2=1-b^2/a^2$。E没有初等函数的理论解析表达式。

例11.2 用数值积分求误差函数

误差函数erf (x)

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \mathrm{e}^{-z^2}\, \mathrm{d}z \tag{11-3}$$

常出现在统计应用和一些抛物线类的微分方程的解中。方程（11-3）的积分也不能通过解析的办法求值。

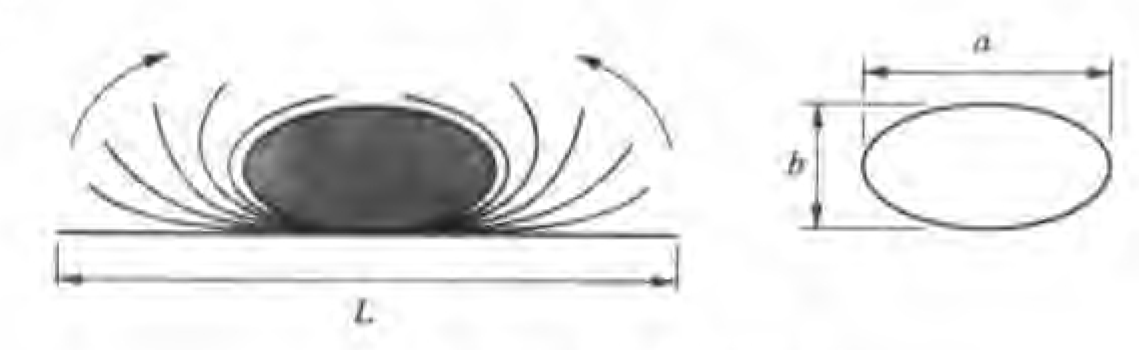


图11-2 用薄板来制造椭圆管道

599

例11.3 多项式积分

多项式$P_n(x) = c_1x^n+c_2x^{n-1} + \dots + c_{n+1}$的定积分为

$$\int_a^b [c_1 x^n + c_2 x^{n-1} + \cdots + c_{n+1}]\mathrm{d}x = \left[\frac{c_1}{n+1}x^{n+1} + \frac{c_2}{n}x^n \cdots c_{n+1}x\right]_{x=a}^{x=b}$$

在MATLAB中，用`polyval`函数可以很容易地由多项式的系数计算出积分（见练习2）。当然，在$P_n(x)$的积分中没有逼近问题，所以本章中会用$P_n(x)$来逼近形如方程（11-1）的$f(x)$。

11.1 基本思想和术语

数值积分也叫数值求积（*numerical quadrature*）。术语"求积"的本意是"求与某个平面图形有相同面积的正方形的边长"。这表明了数值积分的一个基本的计算策略：求$I=\int f(x)\mathrm{d}x$时，用易于积分的简单函数来逼近曲线$y = f(x)$。简单曲线下面的面积近似等于$f(x)$下面的面积。

例如，图11-3描述了对函数的分段线性逼近。近似面积可以通过曲线的分段逼近和x轴之间的梯形面积相加来计算。多项式易于积分，多项式插值的理论也非常简单，因此大部分数值积分方法都是先对$f(x)$构造多项式插值式，然后再对插值式积分来得到$f(x)$的近似积分。

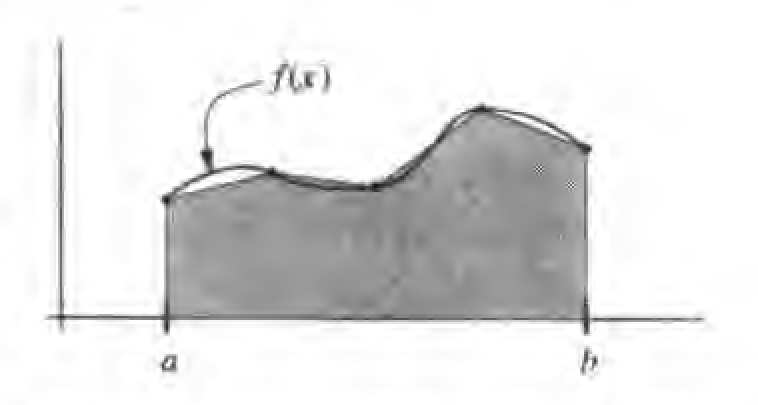


图11-3 $\int_a^b f(x)\,\mathrm{d}x$ 的值用$f(x)$的分段线性插值式下面的阴影面积来逼近

在积分所限定的区间中，被积函数$f(x)$的插值式在n个点处求值，这些点称为节点（node）。节点用x_i表示，并假设它们有序且各不相同（即，$a \leqslant x_1 < x_2 < \dots < x_n \leqslant b$）。如果$x_1 = a$且$x_n = b$，那么区间就是闭的，用符号$[a,b]$表示；如果$x_1>a$且$x_n<b$，那么区间就是开的，用符号$(a,b)$表示。

600

在第10章中我们已经知道，分段多项式的插值比单个多项式的全局插值更加优秀，这个结论对数值积分方法也适用。图11-4表示将整个闭区间$[a,b]$划分为N个小段[⊖]（panel）。在每个小段上对$f(x)$进行低阶多项式逼近。对每个小段上的逼近多项式积分时，就得到基本公式(basic rule)。基本公式只涉及用足够的$(x, f(x))$对来定义分段多项式的某一段，将此公式应用到N个小段并把结果相加就得到了复合公式（composite rule），或称为扩展公式（extended rule）。

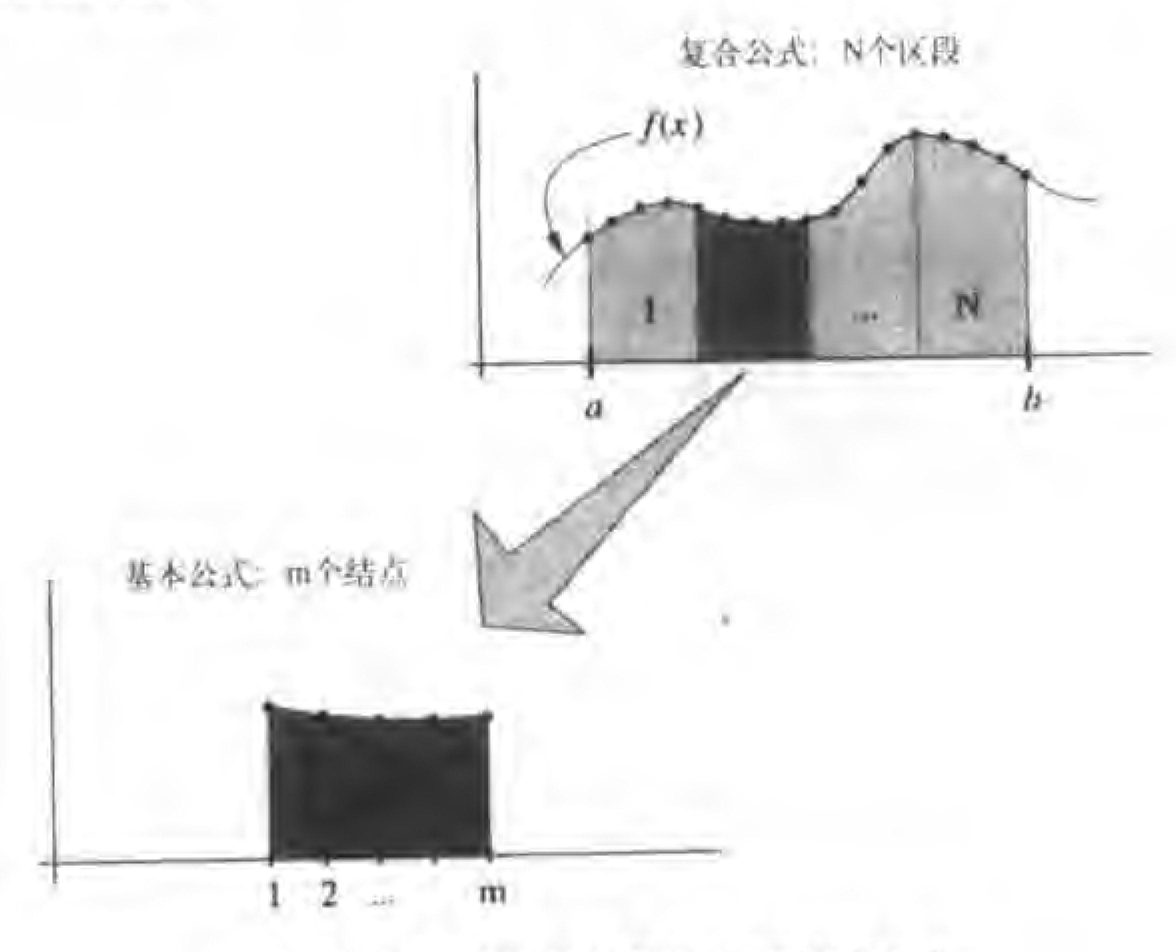


图11-4 $f(x)$数值求积的基本公式和复合公式

在一个小段中节点的位置和数目决定了基本公式的很多重要特性。当节点均匀分布时，所用的积分公式就叫做Newton-Cotes公式。相反，高斯求积公式要选择作为正交多项式零点的节点。高斯求积公式的截断误差比相同数目节点的Newton-Cotes公式的要小得多。虽然高斯求积公式难于推导，但是在程序中实现时不会有明显的困难。

601

截断误差在本章中的角色非常重要。求积公式的截断误差可通过应用于已知解析式的积分来测得。截断误差是数值结果与分析结果之间的差，其理论公式也可以通过数值分析得到。这些公式表明，每种方法的截断误差精度都会随着积分节点数的增加而减小。将测量误差和理论公式作误差比较，可用来验证数值积分法的代码可实现性。

⊖ 在函数quad8的在线帮助中，一个小段就是节点间的距离，而非子区间的宽度。

自适应（*adaptive*）数值积分法通过估计截断误差来决定是否需要更密的分布节点，从而达到指定精度。若需要增加节点，函数会在新节点处重新求值，截断误差也要重新估计。积分范围的自适应细分不只提供了用户自定义容差（tolerance）内的结果，还节省了$f(x)$求值的次数。同时我们也可以预料到，自适应方法会需要更复杂的程序逻辑。

符号和数值积分

本书中，我们主要集中讨论数值计算——对其他方法不能计算（或不方便计算）的某些量进行近似求值。符号计算是另外一种方法，用来运算那些精确表达相关量的解析表达式。能进行符号积分计算的计算机程序有很多。通常，可使用这样的符号计算来得到积分的理论表达式。这样，符号计算工具就充当了创建积分表的角色。Mathworks出售可选的符号数学工具箱(Symbolic Mathematics Toolkit)[⊖]，它可以用来进行积分的符号求值。

例11.4 用MATLAB进行符号积分

如果我们使用的是MATLAB的学生版，或者你的计算机上装了符号数学工具箱，就可以用`int`函数来进行定积分和不定积分的符号计算。函数`int`可以通过以下四种方法调用：

```
I = int(f)
I = int(f,v)
I = int(f,a,b)
I = int(f,v,a,b)
```

其中`f`是用来定义被积函数的符号表达式。`int`函数的前两个调用形式用来计算不定积分。第二个调用形式，`int(f,v)`，允许明确地指定积分变量。当被积函数表达式`f`中包含多于一个符号变量时，指定积分变量是非常必要的。函数`int`的后两种调用形式用来在闭区间$[a,b]$上计算定积分。 602

在使用`int`函数计算积分之前，出现在被积函数中的任何一个变量必须明确地指定为某个符号，这可由函数`sym`或`syms`来实现。例如，定义`x`、`y`和`z`为符号变量，我们可以用

```
>> x = sym('x'),  y = sym('y'),  z = sym('z')
```

或缩写形式

```
>> syms x y z
```

来实现。于是积分

$$I = \int_a^b (x^3 - c)\,dx$$

就可用以下MATLAB语句求值：

```
>> syms x a b c
>> I = int(x^3-c,x,a,b)

I =

1/4*b^4-c*b-1/4*a^4+c*a
```

关于`int`函数的更多信息，可在命令行中键入`help sym/int`，或参考MATLAB的学生版文档或者符号数学工具箱的文档。

⊖ 在MATLAB学生版中包含它的限定版本。

11.2 Newton-Cotes公式

Newton-Cotes积分公式可通过在n个等距点上用$n-1$阶多项式$P_{n-1}(x)$对$f(x)$进行插值来逼近方程（11-1）中的被积函数，并对逼近函数求积分来获得，其一般形式将在11.2.3节中推导。

11.2.1 梯形公式

梯形公式使用线性插值来逼近$f(x)$。如图11-5所示，令$f_1 \equiv f(x_1)$，$f_2 \equiv f(x_2)$。图中区间的一

603

阶（线性）拉格朗日插值多项式是

$$P_1(x) = \frac{x - x_2}{x_1 - x_2} f_1 + \frac{x - x_1}{x_2 - x_1} f_2 \tag{11-4}$$

用$P_1(x)$来逼近$f(x)$，我们可以用下式来估算$f(x)$从x_1到x_2的积分：

$$I = \int_{x_1}^{x_2} f(x)\mathrm{d}x \approx \int_{x_1}^{x_2} P_1(x)\mathrm{d}x$$

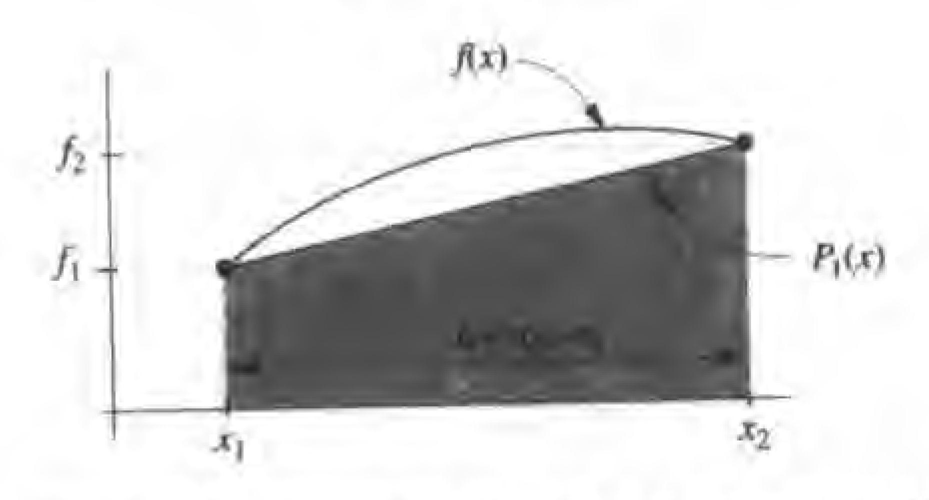


图11-5 使用梯形公式对区间上的两个节点进行数值积分

用方程（11-4）代替积分式中的$P_1(x)$并简化，得

$$\begin{aligned}\int_{x_1}^{x_2} P_1(x)\,\mathrm{d}\,x &= \int_{x_1}^{x_2}\left(\frac{x-x_2}{x_1-x_2}f_1 + \frac{x-x_1}{x_2-x_1}f_2\right)\mathrm{d}\,x \\ &= -\frac{f_1}{h}\int_{x_1}^{x_2}(x-x_2)\,\mathrm{d}\,x + \frac{f_2}{h}\int_{x_1}^{x_2}(x-x_1)\,\mathrm{d}\,x \\ &= \frac{1}{2}(f_1+f_2)h\end{aligned}$$

其中$h \equiv x_2 - x_1$。因此，

$$I = \int_{x_1}^{x_2} f(x)\,\mathrm{d}\,x \approx \frac{1}{2}(f_1+f_2)h \tag{11-5}$$

进行误差分析（例见[10，70]），可知基本梯形公式的截断误差为$O(h^3)$

$$I = \int_{x_1}^{x_2} f(x)\,\mathrm{d}\,x = \frac{1}{2}(f_1+f_2)h + \mathrm{C}h^3 f''(\xi) \tag{11-6}$$

604

其中C是常数，ξ是区间$x_1<\xi<x_2$中的某一点，$f''(x) = \mathrm{d}^2 f / \mathrm{d}x^2$。

复合梯形公式 复合梯形公式是指将积分区间划分为几个子区间或小段(panel)，然后对每个小段应用基本公式求出积分。图11-6描述了复合梯形公式。整个区间的积分值通过将每个小段上积分值相加得到：

$$\int_a^b f(x)\,dx = \int_{x_1}^{x_2} f(x)\,dx + \int_{x_2}^{x_3} f(x)\,dx + \cdots + \int_{x_{n-1}}^{x_n} f(x)\,dx$$

令x_i为闭合区间$[a,b]$上的等距点序列：

$$x_i = a + (i-1)h，其中 i = 1,\ldots,n, \qquad h = \frac{b-a}{(n-1)} \tag{11-7}$$

将方程（11-6）应用于图11-6中的每一个小段，得

$$\begin{aligned}\int_{x_1}^{x_n} f(x)\,dx = &\frac{1}{2}(f_1+f_2)h + C_1h^3f''(\xi_1)\\ &+\frac{1}{2}(f_2+f_3)h + C_2h^3f''(\xi_2)\\ &\vdots\\ &+\frac{1}{2}(f_{n-1}+f_n)h + C_{n-1}h^3f''(\xi_{n-1})\end{aligned}$$

其中$x_i<\xi_i<x_{i+1}$。如果$K = \max[C_if''(\xi_i)]$是有限的，那么未知截断误差和的最大界限就是$(n-1)Kh^3$，且

605

$$(n-1)Kh^3 = (n-1)K\left(\frac{b-a}{n-1}\right)^3 = K\frac{(b-a)^3}{(n-1)^2} = K'h^2$$

其中$K' = K(b-a)$，因此复合梯形公式就是

$$\int_{x_1}^{x_n} f(x)\,dx = h\left[\frac{1}{2}f_1 + f_2 + f_3 + \cdots + f_{n-2} + f_{n-1} + \frac{1}{2}f_n\right] + O(h^2) \tag{11-8}$$

给定被积函数和积分极限，常数K'就是固定的。截断误差中惟一可由用户控制的就是节点数n。

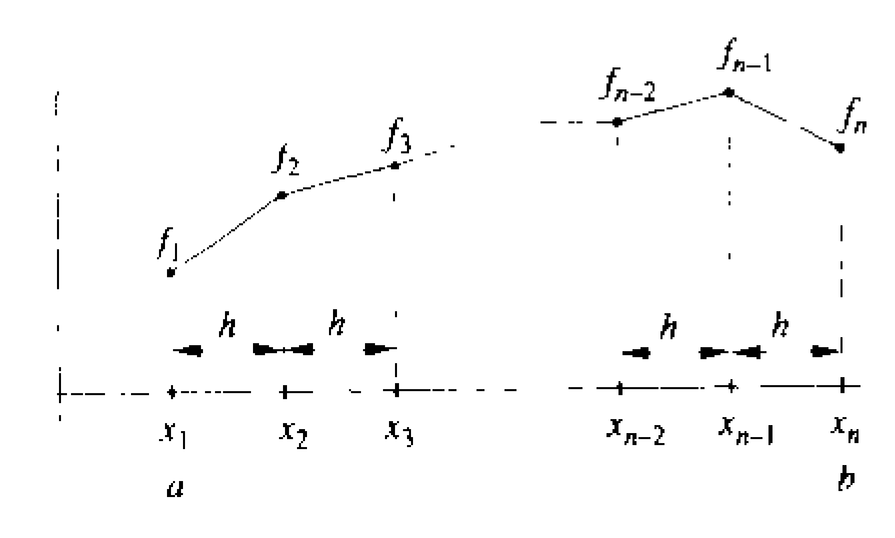


图11-6 梯形公式应用于被n个点划分成$n-1$个小段的积分区间

例11.5 用梯形公式进行手工计算

使用梯形公式来计算erf (1)：

$$I = \mathrm{erf}(1) = \frac{2}{\sqrt{\pi}}\int_0^1 e^{-x^2}\,dx$$

设$n = 5$，步长为$h = (1-0)/4 = 0.25$。下表列出了$\exp(-x_i^2)$在$x_i = (i-1)h$的值：

x_i	0	0.25	0.50	0.75	1.00
$\exp(-x_i^2)$	1.0000	0.9394	0.7788	0.5698	0.3679

将复合梯形公式的各项相加得

$$I = \frac{2}{\sqrt{\pi}}(0.25)[(0.5)(1.0) + 0.9394 + 0.7788 + 0.5698 + (0.5)(0.3679)]$$

$$= \frac{2}{\sqrt{\pi}}(0.7430)$$

$$= 0.8384$$

取四位小数位，由erf (1)的精确值为0.8427，可知以$n=5$逼近的梯形公式的误差为－0.0043。

任意f(x)的梯形公式实现 程序清单11-1中的trapezoid函数使用复合梯形公式对用户自定义函数进行数值积分。这里首先概要介绍trapezoid函数的输入和输出参数，然后讨论函数中的代码。函数trapezoid的调用方法为

```
I = trapezoid(fun,a,b,npanel)
```

其中*fun*是求被积函数的m文件的名字，*a*和*b*是积分的上界和下界，*npanel*是积分中所使用小段的数目。数值积分的精度受*npanel*的影响很大，所以一般在应用trapezoid时要尝试多个*npanel*值来调用函数以使积分值落在期望的容差内。此程序将在例11.6中加以示范。

606

程序清单11-1 函数trapezoid使用梯形公式计算定积分

```
function I = trapezoid(fun,a,b,npanel)
% trapezoid  Composite trapezoid rule
%
% Synopsis:  I = trapezoid(fun,a,b,npanel)
%
% Input:     fun    = (string) name of m-file that evaluates f(x)
%            a, b   = lower and upper limits of the integral
%            npanel = number of panels to use in the integration
%                     Total number of nodes = npanel + 1
%
% Output:    I = approximate value of the integral from a to b of f(x)*dx

n = npanel + 1;       %  total number of nodes
h = (b-a)/(n-1);      %  stepsize
x = a:h:b;            %  divide the interval
f = feval(fun,x);     %  evaluate integrand

I = h * ( 0.5*f(1) + sum(f(2:n-1)) + 0.5*f(n) );
```

参数*fun*指定的m文件必须接受单个的向量x作为输入，并返回单个向量$y=f(x)$。本m文件中进行计算的方法是通常使用的数组操作符。

函数trapezoid的代码很简单。注意，输入参数是小段总数*npanel*，而不是复合公式中使用数据点的总数n。为计算复合梯形公式中的和，方程（11-8）写成

$$I = h\left[\frac{1}{2}f_1 + \sum_{i=2}^{n-1} f_i + \frac{1}{2}f_n\right]$$

直接转变成单行MATLAB代码是:

```
I = h*(0.5*f (1) + sum(f (2:n-1)) + 0.5*f (n));
```

函数trapezoid的应用将在下例中进行示范。

例11.6 用梯形公式求 $\int_0^5 xe^{-x}\,dx$

[607]

本例示范如何用trapezoid函数来逼近下列积分式的值：

$$I=\int_0^5 xe^{-x}\,dx=-e^{-x}(1+x)\Big|_0^5=1-6e^{-5}$$

因为积分的精确值已知，由代码得到的截断误差可以与理论结果相比较。

程序清单11-2的下部分所示的函数xemx可以用来求解$f(x)=xe^{-x}$。数组操作符.*允许输入变量x为向量或标量。如果xemx用在本章介绍的任何数值积分程序中，这就变得至关重要了。

程序清单11-2 函数demoTrap和xemx使用梯形公式求 $\int_0^5 xe^{-x}\,dx$

```
function demoTrap
% demoTrap  Use composite trapezoidal rule to integrate x*exp(-x) on [0,5]
%
% Synopsis:  demoTrap
%
% Input:     none
%
% Output:    Table of integral values for increasing number of intervals

a = 0;  b = 5;   Iexact = -exp(-b)*(1+b) + exp(-a)*(1+a);
fprintf('\n\tIexact = %14.9f\n',Iexact);

fprintf('\n    n      h          I              error         alpha\n');
errold = [];
for np = [ 2 4 8 16 32 64 128 256]
  I = trapezoid('xemx',a,b,np);
  err = I - Iexact;
  n = np + 1;    h = (b-a)/(n-1);      %  number of nodes and stepsize
  fprintf(' %4d %9.5f %14.9f %14.9f',n,h,I,err);

  if ~isempty(errold)
    fprintf('  %9.5f\n',log(err/errold)/log(h/hhold));
  else
    fprintf('\n');
  end
  hhold = h;  errold = err;           %  prep for next stepsize
end

function y = xemx(x)
% xemx  Evaluate x*exp(-x), where x is a scalar or vector
y = x.*exp(-x);
```

[608]

除了使用外部函数如xemx来定义被积函数，还可以使用内嵌函数对象（另见3.6.4节）。下面的语句定义了一个内嵌函数对象然后将它传给trapezoid函数来计算积分：

```
>> f = inline('x.*exp(-x)')
f =
     Inline function:
     f(x) = x.*exp(-x)

>> trapezoid(f,0,5,4)
ans =
    0.8355
```

注意，向另一个函数传递内嵌函数对象时，对象的名字两边不用引号。比较下列语句

```
>> trapezoid('xmex',0,5,4)
ans =
   0.8355
```

它将文件xemx.m中的外部函数传给trapezoid。

程序清单11-2还描述了demoTrap函数，它使用trapezoid和xemx函数在一系列递减的h（递增的n）上用复合梯形公式来求$\int_0^5 xe^{-x}\,dx$的近似值。对每个h，打印出积分的近似值和绝对误差。运行函数demoTrap输出如下：

```
>> demoTrap

Iexact =     0.959572318

   n        h            I              error          alpha
   3    2.50000     0.555143410    -0.404428908
   5    1.25000     0.835474884    -0.124097434      1.70441
   9    0.62500     0.926771814    -0.032800504      1.91968
  17    0.31250     0.951254724    -0.008317594      1.97948
  33    0.15625     0.957485472    -0.002086846      1.99484
  65    0.07812     0.959050139    -0.000522179      1.99871
 129    0.03906     0.959441744    -0.000130574      1.99968
 257    0.01953     0.959539673    -0.000032645      1.99992
```

这里要简要介绍一下alpha列的含义。从输出结果可以看到，误差随着n的增加而减小；有的读者就可能会认为trapezoid函数是正确的，然而，对正确性更仔细的检验还需要检查截断误差的量化特性。

609

方程（11-8）说明梯形公式的截断误差是$O(h^2)$。虽然这不能指出每个h值的截断误差绝对大小，但是可以用数量级估计来预测截断误差随着h的减小而减小的速度。令$E(h)$为给定h值的任何积分值的截断误差。两个h值上E的比率是

$$\frac{E(h_2)}{E(h_1)}=\left(\frac{h_2}{h_1}\right)^{\alpha}$$

解出α，得

$$\alpha=\frac{\log\left(\dfrac{E(h_2)}{E(h_1)}\right)}{\log\left(\dfrac{h_2}{h_1}\right)}$$

当h接近0时，用此公式计算出α值应该接近梯形公式得到的值2。

函数trapezoid输出的最后一列是用前面的公式计算出的α值。随着n的增加，α值接近于2，这大大表明trapezoid函数正确实现了方程（11-8）。但是，一个小小的代码错误就会破坏截断误差正确的状态（见练习7）。

图11-7用图形描述了$n = 5$、7和9时用梯形公式对$\int_0^5 xe^{-x}\,dx$的积分。随着n的增加，梯形片段下面的区域面积的和逐渐接近曲线xe^{-x}下面的区域面积。图示由plotTrapInt函数生成，此函数在NMM工具箱中，此处未列出。

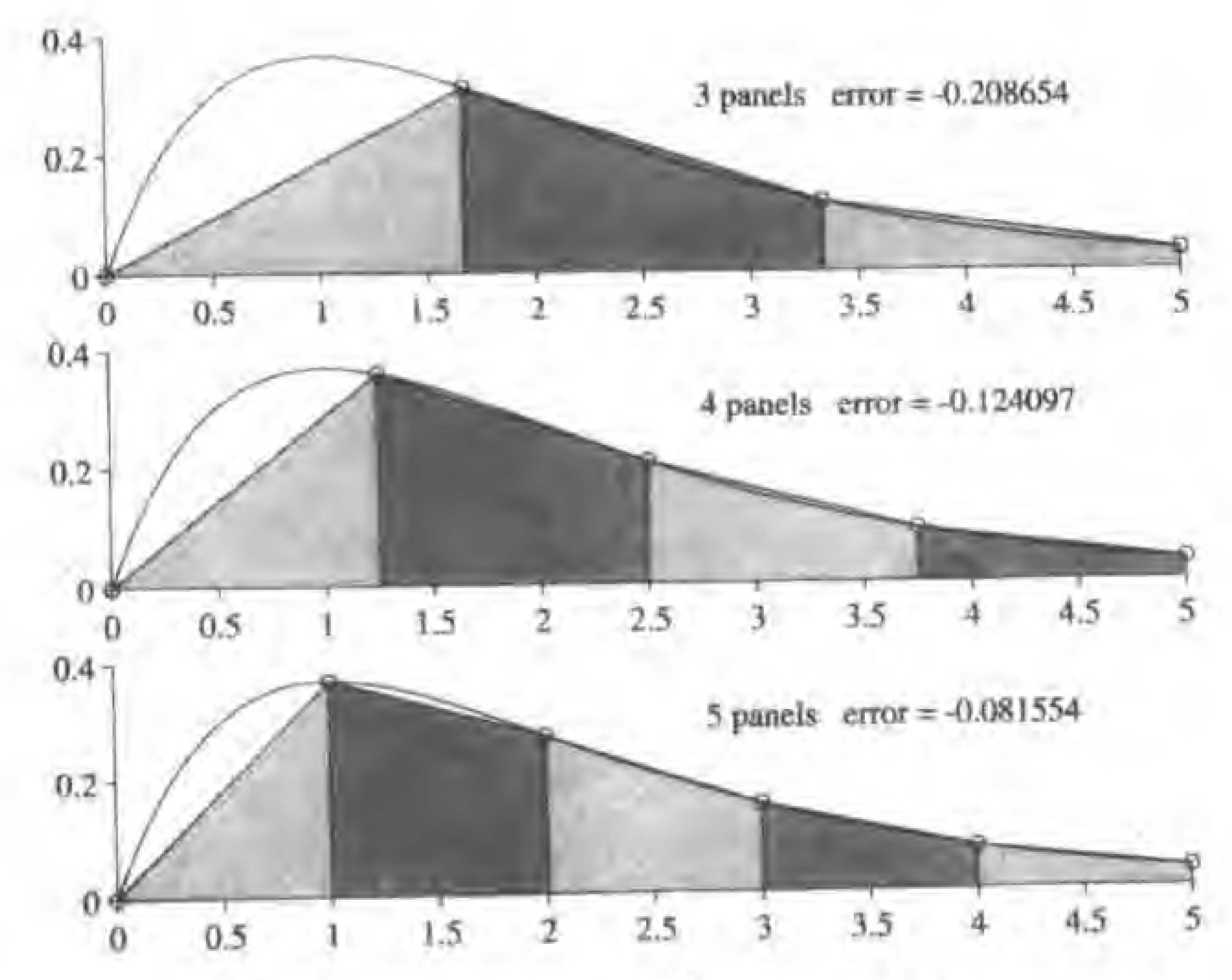


图11-7 随着点数增加梯形公式对 $\int_0^5 x e^{-x} \mathrm{d}x$ 的逼近所得图形



离散数据的梯形公式实现 考虑对从实验中得到的函数$y=f(x)$进行积分。$f(x)$不能在任意的x上求值，因为函数只在有限数据集(x_i,y_i)上存在，$i=1,\ldots,n$。程序清单11-3中的函数`trapzDat`使用梯形公式对这种函数进行积分。除了`trapzDat`函数，内置`trapz`函数也有此功能。

程序清单11-3 `trapzDat`函数使用梯形公式对离散数据积分

```
function I = trapzDat(x,f)
% trapzDat  Composite trapezoid rule for arbitrarily spaced discrete data
%
% Synopsis:  I = trapzDat(x,f)
%
% Input:     x = vector of independent variable data.  Assumed
%                to be unequally spaced
%            f = vector discrete function values to be integrated.
%
% Output:    I = integral of f with respect to x

n = length(f);
if length(x)~=n
  error('Dimensions of x and f are incompatible');
end

dx   = diff(x);                    %  vector of x(i+1)-x(i) values
avef = f(1:n-1) + 0.5*diff(f);     %  vector of average f values
I = sum(avef.*dx);                 %  avef(1)*dx(1) + avef(2)*dx(2) + ...
```

为得到最大的灵活性，函数trapzDat没有假定定义函数$f(x)$的x值是等距分布的。对于不均匀的h，方程（11-8）变成

611

$$\int_{x_1}^{x_n} f(x)\,dx = \sum_{i=1}^{n-1} \frac{1}{2}(f_i + f_{i+1})(x_{i+1} - x_i)$$

trapzDat中的计算只有三行MATLAB代码：

```
dx   = diff(x);                  % vector of x(i+1)-x(i) values
avef = f(1:n-1) + 0.5*diff(f);   % vector of average f values
I = sum(avef.*dx);               % avef(1)*dx(1) + avef(2)*dx(2) + ...
```

向量avef由基于如下恒等式的向量化代码定义：

$$\frac{1}{2}(f_i + f_{i+1}) = f_i + \frac{1}{2}(f_{i+1} - f_i)$$

函数trapzDat的应用在第4章的4.2.2节的例4.3中作了示范，用来测试trapzDat的函数testTrapzDat在第4章的4.2.3节的例4.4中进行了推导。

11.2.2 Simpson公式

Simpson公式使用二阶插值多项式来逼近$f(x)$，其高阶多项式数值积分的截断误差比同样条件下梯形公式的截断误差要小。

如图11-8所示，令x_1，x_2和x_3为x轴上三个等距点，令f_1，f_2和f_3为$f(x)$在这些点上的值。穿过这些点的二阶拉格朗日多项式可以写成

612

$$P_2(x) = \frac{1}{2h^2}(x - x_2)(x - x_3)f_1 - \frac{1}{h^2}(x - x_1)(x - x_3)f_2 + \frac{1}{2h^2}(x - x_1)(x - x_2)f_3 \tag{11-9}$$

其中$h = (x_3 - x_1)/2$。

使用$P_2(x)$插值$f(x)$，对$f(x)$的积分的逼近为

$$I = \int_{x_1}^{x_3} f(x)\,dx \approx \int_{x_1}^{x_3} P_2(x)\,dx = h\left(\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right) \tag{11-10}$$

由误差分析（例见[10、70]）可得

$$I = h\left(\frac{1}{3}f_1 + \frac{4}{3}f_2 + \frac{1}{3}f_3\right) + Ch^5 f^{(4)}(\xi) \tag{11-11}$$

其中C是常数，$f^{(4)} = d^4f/dx^4$，ξ是区间$x_1<\xi<x_3$上的某一点。注意，方程（11-11）中的C不是方程（11-6）中的C。

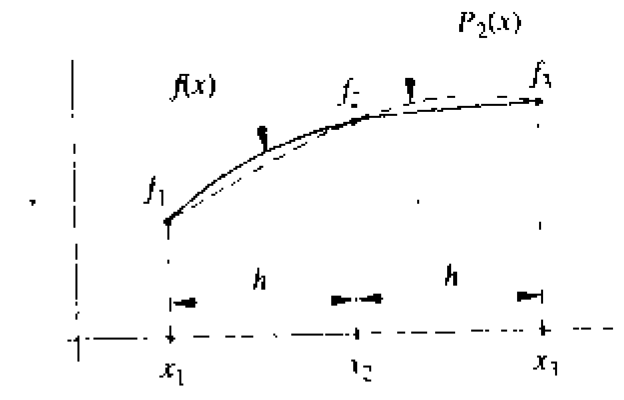


图11-8　Simpson公式应用在一个三节点的小段上。插值多项式$P_2(x)$（虚线）和真实$f(x)$（实线）一般不匹配

复合Simpson公式　图11-9描述了任意长度的区间x划分为$n-1$个等长子区间（小段）的情况，每个子区间的长度为$h = (b-a)/(n-1)$。方程（11-11）中的基本公式要求小段包括三个节点（两个子区间）。复合公式由每个小段的基本公式相加得到。为此，必须要有偶数个宽度为h的子区间。复合Simpson公式要求$n-1$是偶数，故节点总数n为奇数。

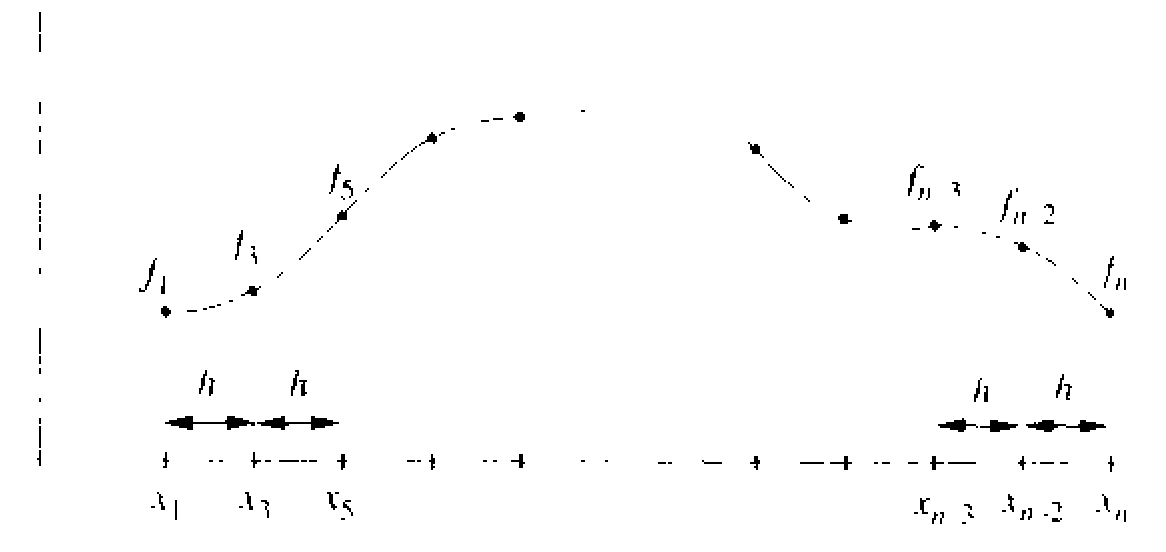


图11-9 Simpson公式应用于一个被n个点划分为$(n-1)/2$个小段的区间

613

将每个小段的基本公式相加，得

$$
\begin{aligned}
\int_{x_1}^{x_n} f(x)\,\mathrm{d}x = & h\left(\frac{1}{3}f_1+\frac{4}{3}f_2+\frac{1}{3}f_3\right)+C_1h^5f^{(4)}(\xi_1) \\
& +h\left(\frac{1}{3}f_3+\frac{4}{3}f_4+\frac{1}{3}f_5\right)+C_2h^5f^{(4)}(\xi_2) \\
& \vdots \\
& +h\left(\frac{1}{3}f_{n-2}+\frac{4}{3}f_{n-1}+\frac{1}{3}f_n\right)+C_{n-1}h^5f^{(4)}(\xi_{n-1})
\end{aligned}
$$

其中$x_i<\xi_i<x_{i+1}$。假设$K=\max[C_i f^{(4)}(\xi_i)]$存在，则复合Simpson公式的截断误差的上界是

$$
(n-1)Kh^5=(n-1)K\left(\frac{b-a}{n-1}\right)^5=K\frac{(b-a)^5}{(n-1)^4}=K'h^4
$$

其中$K'=(b-a)K$。复合Simpson公式变为

$$
\begin{aligned}
\int_a^b f(x)\,\mathrm{d}x = & \frac{h}{3}[f_1+4f_2+2f_3+4f_4+2f_5+\cdots \\
& +2f_{n-4}+4f_{n-3}+2f_{n-2}+4f_{n-1}+f_n]+O(h^4)
\end{aligned}
\tag{11-12}
$$

例11.7 Simpson公式的手工计算

Simpson公式的截断误差比梯形公式的要小得多。本例将证明，对相同的n，Simpson公式对积分的估计会更精确。

在例11.5中，erf (1)的值用$n=5$的梯形公式进行了估计。对相同的数据应用复合Simpson公式，得

$$
\begin{aligned}
I &= \frac{2}{\sqrt{\pi}}\frac{0.25}{3}[1.0+4(0.9394)+2(0.7788)+4(0.5698)+0.3679] \\
&= \frac{2}{\sqrt{\pi}}(0.7469) \\
&= 0.8428
\end{aligned}
$$

$n=5$时Simpson公式的逼近误差为0.0001。对相同数量的$f(x)$的值，梯形公式的误差为 0.0043，显然Simpson公式比梯形公式得到的结果更精确。

任意$f(x)$上Simpson公式的实现 程序清单11-4中的`simpson`函数可以用复合Simpson公

614

式对用户自定义的被积函数积分，其输入输出参数与trapezoid函数的相同。注意，simpson函数的第三个输入参数是小段数量npanel，而不是节点总数n。如果使用n作为输入，按照Simpson公式的要求，就要检查n－1是否为偶数。使用npanel作为输入参数就避免了这种潜在错误和混淆。

程序清单11-4 simpson函数使用Simpson公式求定积分

```
function I = simpson(fun,a,b,npanel)
% simpson   Composite Simpson's rule
%
% Synopsis:  I = simpson(fun,a,b,npanel)
%
% Input:     fun    = (string) name of m-file that evaluates f(x)
%            a, b   = lower and upper limits of the integral
%            npanel = number of panels to use in the integration
%                     Total number of nodes = 2*npanel + 1
%
% Output:    I = approximate value of the integral from a to b of f(x)*dx

n = 2*npanel + 1;     % total number of nodes
h = (b-a)/(n-1);      % stepsize
x = a:h:b;            % divide the interval
f = feval(fun,x);     % evaluate integrand

I = (h/3)*( f(1) + 4*sum(f(2:2:n-1)) + 2*sum(f(3:2:n-2)) + f(n) );
%           f(a)         f_even              f_odd          f(b)
```

复合Simpson公式表达式通过检查项数的奇偶性来决定该项系数是2还是4。方程（11-12）可写成

$$I=\frac{h}{3}\left[f_1+4\sum_{i=2,4,\ldots}^{n-1}f_i+2\sum_{i=3,5,\ldots}^{n-2}f_i+f_n\right]$$

假设离散$f(x)$数据存储在向量f中，方括号内的表达式可以用语句

```
f(1)+ 4*sum(f(2:2:n-1))+ 2*sum(f(3:2:n-2))+ f(n)
```

来求值，其中f(2:2:n-1)是f中下标为偶数的项组成的向量，f(3:2:n-2)是下标为奇数的项组成的向量。

615

例11.8 用Simpson公式求$\int_0^5 xe^{-x}\,dx$

本例使用Simpson公式重复了例11.6中的计算。函数demoSimp使用simpson和xemx函数来求$\int_0^5 xe^{-x}\,dx$，demoSimp在此处未列出，因为它与demoTrap函数几乎完全相同，只是用

```
I = simpson('xemx',xmin,xmax,np);
```

来代替了

```
I = trapezoid('xemx',xmin,xmax,np);
```

运行demoSimp得如下结果：

```
>> demoSimp
```

```
Iexact =      0.959572318

  n       h            I              error            alpha
  3    2.50000    0.712116434    -0.247455884
  5    1.25000    0.928918708    -0.030653610    3.01304
  9    0.62500    0.957204124    -0.002368194    3.69420
 13    0.41667    0.959084588    -0.000487730    3.89706
 17    0.31250    0.959415695    -0.000156623    3.94852
 65    0.07812    0.959571695    -0.000000623    3.98702
129    0.03906    0.959572279    -0.000000039    3.99870
257    0.01953    0.959572316    -0.000000002    3.99967
```

正如预想的那样，误差比梯形公式产生的要小得多。另外，截断误差阶数α在n增加时接近4，这也与方程（11-12）一致。

图11-10描述了用n =7，9和11的Simpson公式计算$\int_0^5 xe^{-x}\,dx$的结果。图中的曲线由`plotSimpInt`函数产生，此函数包含在NMM工具箱中，此处未列出。

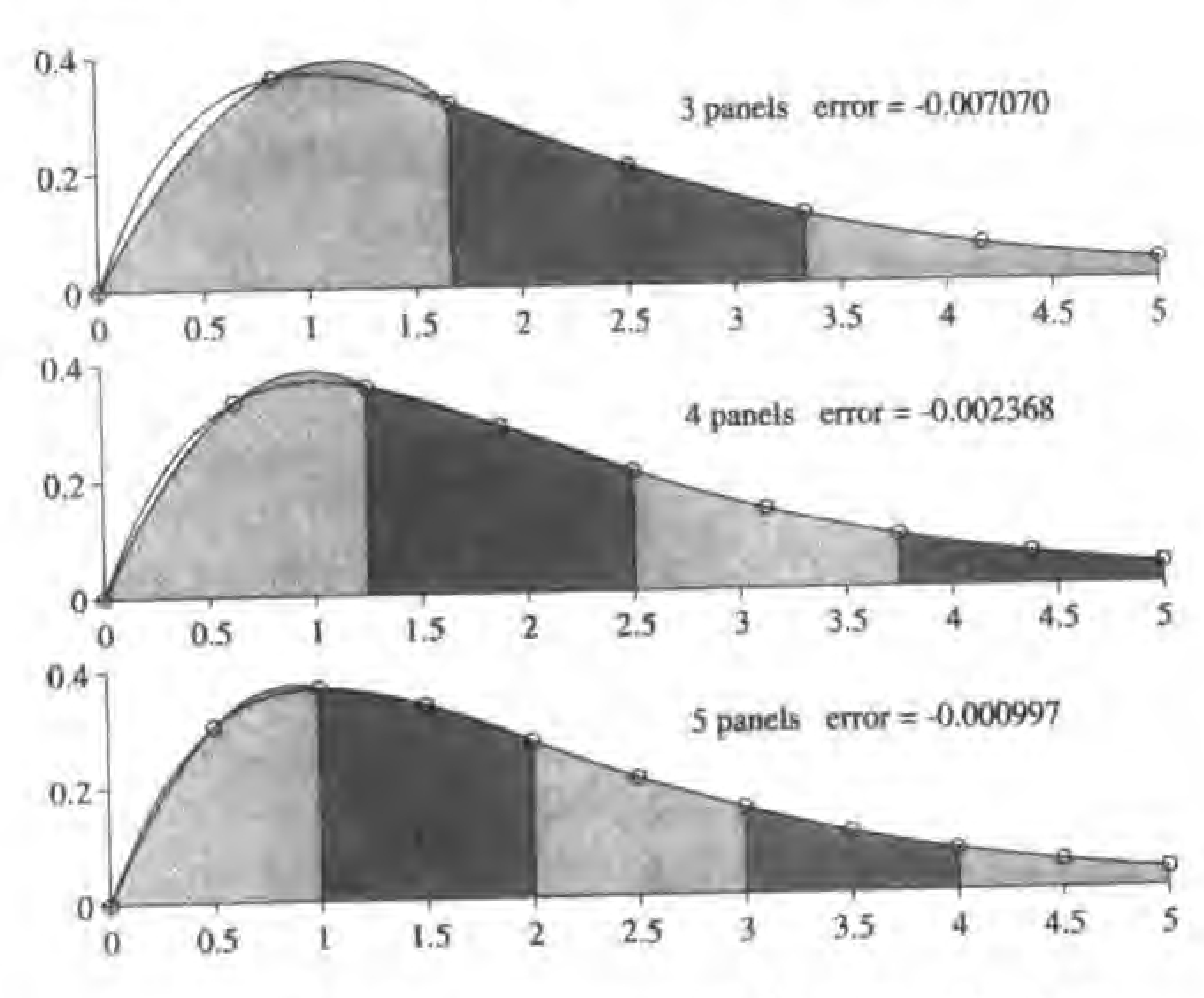


图11-10 随着点数增加，使用Simpson公式对$\int_0^5 xe^{-x}\,dx$逼近的曲线图

11.2.3 Newton-Cotes公式总览

对定积分

$$I=\int_a^b f(x)\,dx$$

经替换$f(x)=P_{n-1}(x)$，然后求替换多项式的积分，可得到对该定积分逼近的基本公式。使用$P_{n-1}(x)$的拉格朗日插值式表示，上式为: [616]

$$\begin{aligned}\int_a^b P_n(x)\,\mathrm{d}x &= \int_a^b [L_1(x)f_1 + L_2(x)f_2 + \cdots + L_n(x)f_n]\,\mathrm{d}x \\ &= \left[\int_a^b L_1(x)\,\mathrm{d}x\right]f_1 + \left[\int_a^b L_2(x)\,\mathrm{d}x\right]f_2 + \cdots + \left[\int_a^b L_n(x)\,\mathrm{d}x\right]f_n \\ &= c_1 f_1 + c_2 f_2 + \cdots + c_n f_n \\ &= \sum_{i=1}^{n} c_i f_i\end{aligned}$$

其中c_i称为权系数，$f_i = f(x_i)$是被积函数在区间$a \leqslant x_i \leqslant b$内$x_i$点处求得的函数值。由前面的推导可知，权系数由下列方程得到：

617

$$c_i = \int_a^b L_i(x)\,\mathrm{d}x \tag{11-13}$$

其中

$$L_i(x) = \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k}$$

对较大的n，c_i的推导使用一种称为待定系数法（undetermined coefficient）的等价过程（例见[10，13]），它比直接计算方程（11-13）更简便。

如果由

$$I = \sum_{i=1}^{n} c_i f_i$$

给出的Newton-Cotes公式能产生$f(x)$积分的精确值，就说该公式有n阶精度，其中$f(x)$是n阶或更小阶数的多项式。根据基础插值多项式的精确性，n节点求积公式的精度好像不可能大于$n-1$。对Newton-Cotes公式确实如此[⊖]，但是使用n个节点的高斯求积法（见11.3节）就可能得到大于n阶的精度。

下面列出了更高阶Newton-Cotes公式的简表。对更大的列表，可参考Abramowitz和Stegun [1]或Davis和Rabinowitz [13]。本节的剩余部分只关心用最少的点定义数值积分公式的基本公式。对这里介绍的每个基本公式，推导其复合公式是很简单的。这里还列出了每个公式的截断误差的准确公式。其截断误差形式为

$$Ch^{\alpha} f^{(\beta)}(\xi)$$

其中C是常数；α是一个整数，代表求积法阶数；β也是一个整数，它表示导数

$$f^{(\beta)} \equiv \frac{\mathrm{d}^{\beta} f}{\mathrm{d} x^{\beta}}$$

的阶，ξ是区间$a<\xi<b$上的某一点。记住，复合公式的阶数比下列基本公式的阶数小1。

所有的Newton-Cotes公式都使用区间上的等距点，其中闭的公式(closed rule)和开的公式(open rule）的区别只是端点是否包括在节点内。图11-11描述了闭的三节点公式（Simpson公式）和相应的开的三节点公式。闭的公式使用定义在区间$[a,b]$上的n个节点，并有$x_1 = a$和$x_n = b$。使用n个节点的开的公式有$x_1>a$和$x_n<b$。

618

⊖　使用n个节点的Newton-Coted公式并不一定保证有n-1阶的精度。比较图11-11下的Simpson公式和3/8Simpson公式。

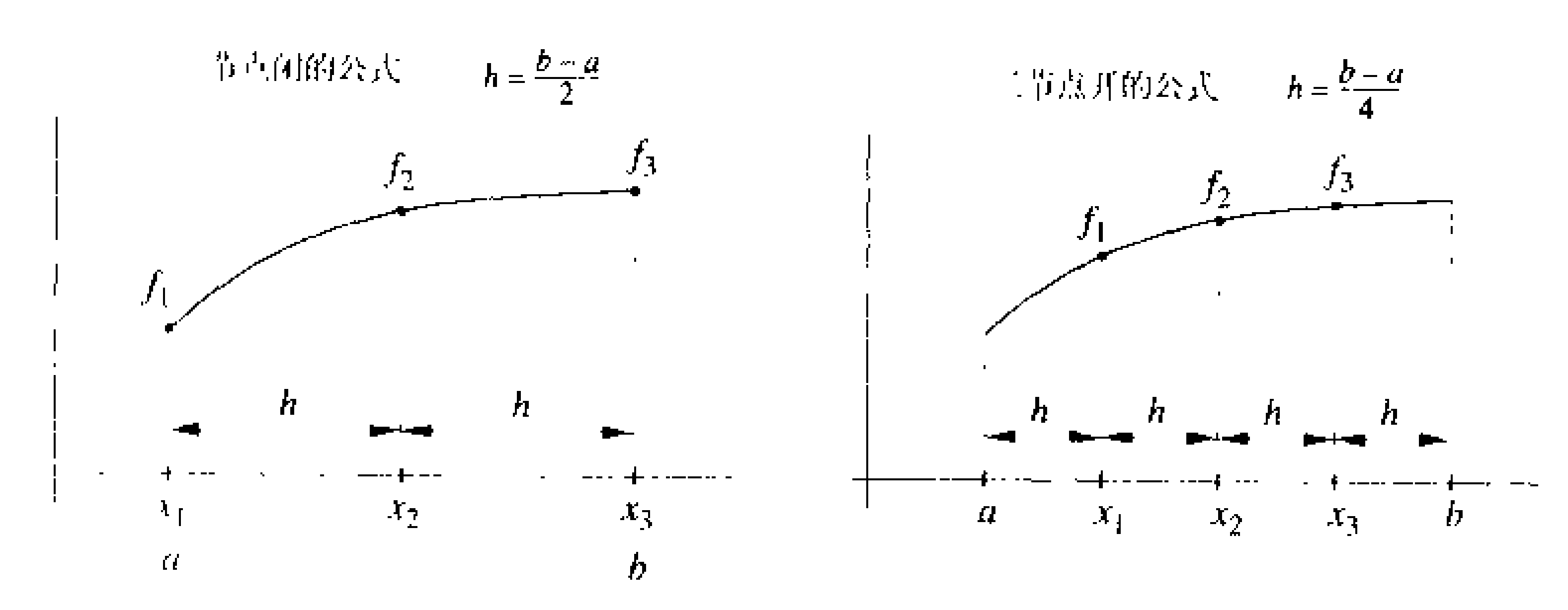


图11-11 闭区间和开区间的三节点Newton-Cotes公式

闭区间公式 对下列每个公式，节点在区间上等距离分布，即

$$x_i = a+(i-1)h$$

梯形公式为：

$$\int_a^b f(x)\,\mathrm{d}x=\frac{h}{2}(f_1+f_2)-\frac{h^3}{12}f''(\xi),\quad h=b-a$$

Simpson公式为：

$$\int_a^b f(x)\,\mathrm{d}x=\frac{h}{3}(f_1+4f_2+f_3)-\frac{h^5}{90}f^{(4)}(\xi),\quad h=\frac{b-a}{2}$$

Simpson $\frac{3}{8}$ 公式为：

$$\int_a^b f(x)\,\mathrm{d}x=\frac{3h}{8}(f_1+3f_2+3f_3+f_4)-\frac{3}{80}h^5f^{(4)}(\xi),\quad h=\frac{b-a}{3}$$

六节点公式为：

$$\int_a^b f(x)\,\mathrm{d}x=\frac{5h}{288}(19f_1+75f_2+50f_3+50f_4+75f_5+19f_6)-\frac{275}{12096}h^7f^{(6)}(\xi),\qquad h=\frac{b-a}{5}$$

619

九节点公式为：

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x=&\frac{h}{14175}(3956f_1+23552f_2-3712f_3+41984f_4\\&-18160f_5+41984f_6-3712f_7+23552f_8+3956f_9)\\&-\frac{2368}{467775}h^{11}f^{(10)}(\xi)\\&h=\frac{b-a}{8}\end{aligned}\tag{11-14}$$

注意，九节点公式中有的权为负值。这也许是一个缺点，因为负数权可能会抵消掉恒正函数$f(x)$的舍入误差。其他的高阶Newton-Cotes公式也有负值权。

开区间公式 开的公式在积分计算上稍微有一些优势，它在端点处有弱奇异（mild sigularity）。特别地，如果$f(x)$在a或b处无定义（或无穷大），且 $f'(x)$ 在端点处有界，那么开的

公式可能会得到很好的逼近结果，而闭的公式就不能，但是开的公式并非万能的（见[13,P71]）。Press等[61]建议避免开的公式，相反，他建议仅仅在区间两端的小段上使用“半开”公式，而且这些公式要有助于避免轻微奇异。

三节点开的公式为：

$$\int_a^b f(x)\,\mathrm{d}x = \frac{4h}{3}(2f_1 - f_2 + 2f_3) + \frac{14h^5}{45}f^{(4)}(\xi)$$

$$h = \frac{b-a}{4} \qquad x_i = a + ih$$

注意，这个公式也有负数权值，可能会抵消积分计算中的误差。

11.3　高斯求积法

高斯求积法的基本公式与Newton-Cotes公式所使用的计算公式相同，对n个节点：

620

$$I = \int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i f(x_i) \tag{11-15}$$

高斯求积法通过选择方程（11-15）中特别的节点(x_i)和权(c_i)组合，从而使数值积分的准确性达到最优。所得到的求积公式在如下意义上最优：如果$f(x)$是一个k阶多项式，那么n个节点的高斯求积公式在$k \le 2n-1$的情况下是精确的，这是被积函数在n个节点积分时可能达到的最高精度。高斯求积法对节点和权的选择取决于正交多项式的特性，下节将给出这种理论的概述。虽然高斯求积法的推导比Newton-Cotes公式的推导更棘手，但是高斯求积法的实现并没有复杂多少。在本节的结尾，给出了gaussQuad函数来使用Gauss-Legendre求积法对用户自定义的函数$f(x)$进行数值积分。

在深入研究基本的理论之前，考虑下面的例子，它显示了高斯求积法只需对被积函数进行少量计算就可能得到精确结果。

例11.9　示范高斯求积法的精度

本例提供了erf(1)求值的一种“计算秘诀”。其目的不是硬套公式，而是给那些研究为什么高斯求积法如此有效的人提供动力和希望，之后我们会介绍高斯求积法的理论。

使用两个节点的高斯求积公式，所得的 $\mathrm{erf}(x)=(2/\sqrt{\pi})\int_0^x \exp(-z^2)\,\mathrm{d}z$ 的逼近却令人吃惊地精确，此公式如下：

$$I = \int_a^b f(x)\,\mathrm{d}x = \frac{b-a}{2}(c_1 f(x_1) + c_2 f(x_2))$$

其中

$$x_1 = \frac{b+a}{2} - \left(\frac{b-a}{2}\right)\frac{1}{\sqrt{3}},\quad x_2 = \frac{b+a}{2} + \left(\frac{b-a}{2}\right)\frac{1}{\sqrt{3}},\quad c_1 = 1,\quad c_2 = 1$$

注意，积分的近似值形式与Newton-Cotes的一样。最大的不同在于节点x_i的位置表达式形式更复杂，且权c_i的值比较特殊。计算I的近似值，保留7位小数位，得

$$x_1 = \frac{1}{2} - \left(\frac{1}{2}\right)\frac{1}{\sqrt{3}} = 0.2113249,\qquad x_2 = \frac{1}{2} + \left(\frac{1}{2}\right)\frac{1}{\sqrt{3}} = 0.7886571$$

$$f_1 = \exp(-x_1^2) = 0.9563243 \qquad f_2 = \exp(-x_2^2) = 0.5368803$$

$$\mathrm{erf}(1) \approx \frac{2}{\sqrt{\pi}}\frac{1}{2}[(1.0)(0.9563243) + (1.0)(0.5368803)] = 0.8424504$$

近似值有2.5×10^{-4}的误差，比5个节点（不是2个）的Simpson公式产生的误差小100倍。这种异常优越的高精度是对高斯求积公式中的节点和权谨慎选择的结果。 621

11.3.1 理论基础

在高斯求积法的推导中多项式用作$f(x)$的模型。我们希望，如果基本求积公式在对高阶多项式的积分上做的很好，就有可能（但不能保证）会对其他函数的积分逼近得很好。这对Newton-Cotes公式也适用。但是，多项式和高斯求积法之间的这种关系更深一些。

本节将推导高斯求积法的基本原理。虽然我们没有给出正式的证明，但是给出了与之相当的数学推导。此推导的主要步骤如下：

1. 在被积函数多项式插值的基础上构造求积公式截断误差的理论表达式。

2. 当m阶多项式被积函数使用n阶多项式插值时，截断误差可变成绝对零。找出在$m>n$下使截断误差为零的条件。

3. 总结出作为被积函数的最好插值式的正交多项式的特性。

误差表达式 在任意函数$f(x)$的数值积分中首先建立误差表达式是很有用的。根据[11，40]，首先推导$f(x)$多项式插值的误差表达式，然后对插值误差进行积分。

令$P_{n-1}(x)$为在n个点$x_1,\dots,x_n$上对$f(x)$进行插值的多项式，其阶数至多为$n-1$。由Newton插值多项式的定义方程（10-26）知

$$P_{n-1}(x)=f[x_1]+(x-x_1)f[x_1,x_2]+\cdots+(x-x_1)(x-x_2)\cdots(x-x_{n-1})f[x_1,x_2,\dots,x_n]$$

令$\hat{x}$为异于点$x_1,\dots,x_n$的任意点，令$P_n(x)$为在点$x_1,\dots,x_n$以及$\hat{x}$上对$f(x)$进行插值的n阶多项式：

$$P_n(x)=f[x_1]+(x-x_1)f[x_1,x_2]+\cdots+(x-x_1)(x-x_2)\cdots(x-x_n)f[x_1,x_2,\dots,x_n,\hat{x}]$$

622

或者使用方程（10-27）的递归形式，

$$P_n(x)=P_{n-1}(x)+f[x_1,x_2,\dots,x_n,\hat{x}]\prod_{i=1}^{n}(x-x_i)$$

由定义，$P_n(\hat{x})=f(\hat{x})$，故

$$f(\hat{x})=P_{n-1}(\hat{x})+f[x_1,x_2,\dots,x_n,\hat{x}]\prod_{i=1}^{n}(\hat{x}-x_i)$$

且$x=\hat{x}$处插值式$P_{n-1}(x)$的误差为

$$e_{n-1}(\hat{x})=f(\hat{x})-P_{n-1}(\hat{x})=f[x_1,x_2,\dots,x_n,\hat{x}]\prod_{i=1}^{n}(\hat{x}-x_i) \tag{11-16}$$

对$\hat{x}$只有惟一的限制，就是它不能是任何一个定义插值式的节点。这很难说是个限制，因为由插值式的定义，对任意$j=1,\cdots,n$，有$e_{n-1}(x_j)\equiv0$，因此，方程（11-16）给出的是任何异于定义插值式节点集的x上的插值误差。

使用n个节点的$f(x)$的数值积分可通过对$f(x)$的插值式$P(x)$积分得到：

$$\int_a^b f(x)\,\mathrm{d}x\approx\int_a^b P_{n-1}(x)\,\mathrm{d}x$$

于是，$f(x)$的n节点的数值积分误差为：

$$E_n\{f\} = \int_a^b f(x)\,\mathrm{d}x - \int_a^b P_{n-1}(x)\,\mathrm{d}x$$
$$= \int_a^b f[x_1, x_2, \cdots, x_n, x]\prod_{j=1}^{n}(x - x_j)\,\mathrm{d}x \tag{11-17}$$

其中第二行是用方程（11-16）代替$f(x) - P_{n-1}(x)$而得到的。

最高精度　前面提到，数值积分公式的精度是指能被精确积分的多项式的最大阶数。梯形公式的精度是1，Simpson公式的精度是2，高斯求积公式通过选择节点的位置和权值得到最高精度。

[623]

为方便起见，令

$$\psi(x) = (x - x_1)(x - x_2)\cdots(x - x_n)$$

于是方程（11-17）可以写成

$$E_n\{f\} = \int_a^b f[x_1, x_2, \cdots, x_n, x]\psi(x)\,\mathrm{d}x \tag{11-18}$$

如果$f(x)$是阶数为k且$k<n$的多项式，那么有$f[x_1,\cdots,x_n,x] = 0$（见10.2.3节中"均差性质"的性质3），$E_n\{f\} = 0$。它定义了多项式被积函数上的Newton-Cotes公式的精度极限，所以这并非新内容。如果$f(x)$不是一个多项式，则若积分公式中的节点为$\psi(x)$的零点，误差就为零。这是一个临界结果，也是高斯求积公式中选择节点的基础。注意$\psi(x)$不是在积分公式中用来插值$f(x)$的多项式，有关$\psi(x)$的完整说明还有待于确定。

为使$k>n$时也有$E_n\{f\} = 0$，必须对$\psi(x)$附加额外的条件。假设$f(x)$是阶数最大为$n+m$的多项式，那么f的n阶均差就是阶数最大为m的多项式，故对某些系数σ_i有

$$f[x_1, \cdots, x_n, x] = \sum_{i=0}^{m}\sigma_i x^i$$

这些系数的值并不重要。在假设条件下，方程（11-18）变成：

$$E_n\{f\} = \int_a^b f[x_1, \cdots, x_n, x]\,\psi(x)\,\mathrm{d}x$$
$$= \int_a^b \left[\sum_{i=0}^{m}\sigma_i x^i\right]\psi(x)\,\mathrm{d}x$$
$$= \sigma_0\int_a^b \psi(x)\,\mathrm{d}x + \sigma_1\int_a^b x\psi(x)\,\mathrm{d}x + \cdots + \sigma_m\int_a^b x^m\psi(x)\,\mathrm{d}x$$

不管σ的值，只要下式成立

$$\text{对任意 } i = 0, 1, \cdots, m,\ \text{有 } \int_a^b x^i\psi(x)\,\mathrm{d}x = 0 \tag{11-19}$$

就有$E_n\{f\} = 0$。如果方程（11-19）中的条件满足，那么若$f(x)$是阶数为k且$k \leqslant n+m$的多项式，对积分的逼近就非常精确。数值积分能保持精确的最大k值（上限）为$2n-1$。

[624]

正交多项式　两个标量函数$g(x)$和$h(x)$的内积定义为

$$\langle g, h\rangle = \int_a^b w(x)g(x)h(x)\,\mathrm{d}x \tag{11-20}$$

其中$w(x)$称为权函数（weight function）。如果

$$\text{对任意 } g(x) \neq h(x),\ \text{都有 } \int_a^b w(x)g(x)h(x)\,\mathrm{d}x = 0 \tag{11-21}$$

那么这两个函数就是关于权函数正交的。例如，由直接计算可知，只要$w(x)=1$或$w(x)=(1-x^2)^{-1/2}$，$g(x)=1$和$h(x)=x$就在$[a,b]=[-1,1]$上正交（见练习17）。

对每个$w(x)$，存在n阶多项式组$\{\Phi_n(x)\}$，只要

$$\langle \Phi_i(x),\Phi_j(x)\rangle = \int_a^b w(x)\Phi_i(x)\Phi_j(x)\,\mathrm{d}x = 0,\quad i \neq j \qquad (11\text{-}22)$$

多项式就是正交的。

方程（11-19）要求误差项中的多项式$\psi(x)$关于权函数$w(x)=1$正交。因此，若高斯求积公式的节点是一个正交多项式的零点，就有$E_n(f)=0$。由正交多项式的其他特性可以保证，$\psi(x)$的零点定义的节点在对$f(x)$进行插值时是有益的。特别地，实系数正交多项式的零点（根节点）是实数且是单一的（即每个最低点都不同），它们落在所定义区间$[a,b]$的内部（有关例子见[70,定理3.6.10]）。另外，由三项循环关系（three-term recurrence relationship）也可以产生正交多项式组，其形式为：

$$\Phi_n(x) = (\rho_n x + \tau_n)\Phi_{n-1}(x) - \xi_n\Phi_{n-2}(x)$$

它带有标量系数ρ_n、τ_n和ξ_n。

在区间$[-1,1]$上定义的Legendre多项式组有$w(x)=1$。前四个Legendre多项式是

$$\Phi_0 = 1$$
$$\Phi_1 = x$$
$$\Phi_2 = \frac{1}{2}(3x^2 - 1)$$
$$\Phi_3 = \frac{1}{2}(5x^3 - 3x)$$

625

Legendre多项式的循环关系是

$$\Phi_n(x) = \frac{2n-1}{n}x\,\Phi_{n-1}(x) - \frac{n-1}{n}\Phi_{n-2}(x),\quad n \geqslant 2$$

使用Legendre多项式的高斯求积法就叫做Gauss-Legendre求积法。

11.3.2 Gauss-Legendre求积法的基本公式

由上面的叙述，现在可以推导Gauss-Legendre求积法的算法了。节点的位置由n阶Legendre多项式的根给出。这种对节点的选择可使积分公式的精度为$2n-1$。既然Legendre多项式的正交特性只在$-1 \leqslant x \leqslant 1$上存在，那么求积公式就必须表示成带有此范围限制的积分。在11.3.5节中，将开发$-1 \leqslant x \leqslant 1$上定义的基本公式应用于任意有界积分的程序。

n节点的Gauss-Legendre求积公式为

$$\int_{-1}^{1} f(x)\,\mathrm{d}x \approx \int_{-1}^{1} P_{n-1}(x)\,\mathrm{d}x = \sum_{j=1}^{n} c_j f(x_j)$$

其中$P_{n-1}(x)$是在求积公式的n个节点上对$f(x)$的插值。$P_{n-1}(x)$不能也为$n-1$阶的Legendre多项式，因为Legendre多项式在节点（与插值$f(x)$的节点相同）处总为零。插值多项式用拉格朗日基本插值公式表示，而权由Newton-Cotes公式中的相同程序得到（比较方程（11-13））：

$$c_j = \int_{-1}^{1} L_j(x)\,\mathrm{d}x \qquad (11\text{-}23)$$

这与同阶数的Newton-Cotes公式得到的权不相等，因为节点在Legendre多项式的零点处。例11.11示范了$n=2$时计算x_j和c_j的详细过程。

已知求节点和权的过程，就可以推导出算法11-1，该算法用Gauss-Legendre求积法来逼

近 $I=\int_{-1}^{1} f(x)\,\mathrm{d}x$。

算法11-1 Gauss-Legendre求积法的基本公式

```
specify n, the number of nodes
obtain n node locations, x_j, and weights, c_j, for −1 ≤ x ≤ 1
evaluate f(x_j), j = 1,...,n
initialize: I = 0
for j = 1,...,n
   I ← I + c_j f(x_j)
end
```

626

除了求出令I的精度最大的x_j和c_j的部分外，计算I的过程还是很简单的。11.3.4节描述了一种计算任意阶数的x_j和c_j的方法。11.3.5节介绍了Gauss-Legendre求积法如何才能在任意有限的积分中应用。

例11.10 使用Gauss-Legendre求积公式进行手工计算

均值为0、标准方差为1的标准正态随机变量的分布函数为

$$F(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \mathrm{e}^{-z^2/2}\,\mathrm{d}z$$

随机的x落在距平均值为1的标准方差内的概率为

$$F(1)-F(-1)=\frac{1}{\sqrt{2\pi}}\int_{-1}^{1} \mathrm{e}^{-z^2/2}\,\mathrm{d}z$$

此积分的界限是−1和1，因此可以直接用Gauss-Legendre公式求解。

五节点的Gauss-Legendre公式的节点和权为

j	x_j	c_j
1	−0.9061798459386640	0.2369268850561890
2	−0.5384693101056830	0.4786286704993660
3	0.0000000000000000	0.5688888888888890
4	0.5384693101056830	0.4786286704993660
5	0.9061798459386640	0.2369268850561890

计算表格中的各项，得

j	$\exp(-x_j^2/2)$	$c_j\exp(-x_j^2/2)$
1	0.6632648101210487	0.1571452654293647
2	0.8650442658532125	0.4140349868884232
3	1.0000000000000000	0.5688888888888890
4	0.8650442658532125	0.4140349868884232
5	0.6632648101210487	0.1571452654293647

627

$$\sum c_j\exp(-x_j^2/2)=1.7112493935244650$$

于是

$$F(1)-F(-1)=\frac{1.7112494}{\sqrt{2\pi}}=0.6826897$$

11.3.3 查表求节点和权

Gauss-Legendre求积法的节点和权值在很多的参考书中已列成表格，如Abramowitz和Stegun [1]就给出了直到96阶的节点和权值的表格。对较大的n计算节点和权值并非易事，其中11.3.4节就给出了一种计算方法。

图11-12用图形描述了2到6阶Gauss-Legendre求积法的节点和权值。由此图或相应表格，可以看出如下特性：

- 任意n阶的权都是正的。
- 节点关于$x=0$左右对称分布。
- 与$x=0$距离相等的节点其权值相等。

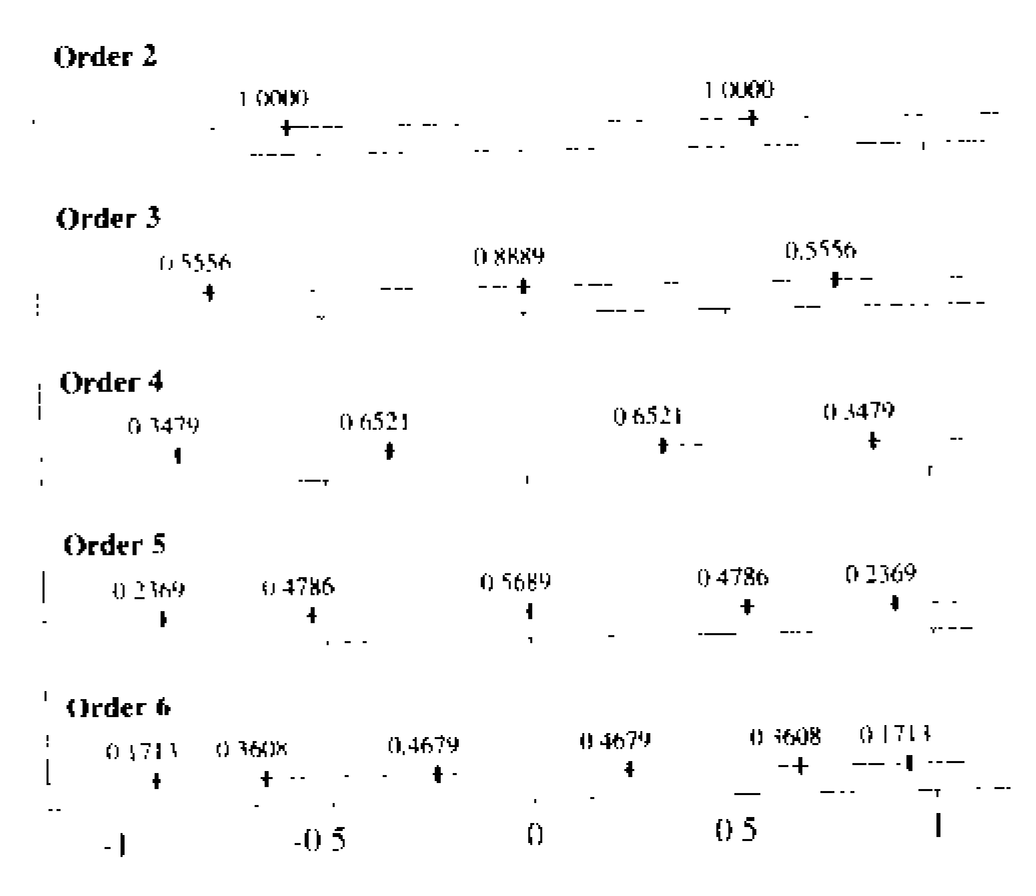


图11-12 2到6阶Gauss-Legendre求积法的节点位置和权值。其中，结点的位置由粗短的垂线来标记，节点标记的上方是相应的权值

628

程序清单11-5中所示的GLtable函数提供了1到8阶Gauss-Legendre求积法的节点位置和权值。其中（几乎所有）的值由赋值得到，而不是计算出来的，所以使用GLtable非常高效。要使用更高阶的公式，可以通过修改GLtable将手册（如文献[1]）中的节点和权值加到GLtable中来实现。另外，也可以使用NMM工具箱中的程序makeGLtable，它将在下节中讨论。

程序清单11-5 函数GLtable返回阶数小于等于8的Gauss-Legendre求积法的节点和权值

```
function [x,w] = GLTable(n)
% GLTable  Nodes and weights for Gauss-Legendre quadrature of order n<=8
%
% Synopsis:  [x,w] = GLTable(n)
%
% Input:     n = number of nodes in quadrature rule, maximum: n = 8
```

```
%
% Output.     x = vector of nodes
%             w = vector of weights

% Numerical values from "Handbook of Mathematical Functions",
% Abramowitz and Stegun, eds., 1965 Dover (reprint), Table 25.4, p. 916

nn = fix(n);                 % Make sure number of nodes is an integer
x = zeros(nn,1);  w = x;     % Preallocate x and w vectors

switch nn
  case 1
    x = 0;   w = 2;

  case 2
    x(1) = -1/sqrt(3);      x(2) = -x(1);
    w(1) =  1;              w(2) =  w(1);

  case 3
    x(1) = -sqrt(3/5);     x(2) =  0;      x(3) = -x(1);
    w(1) =  5/9;           w(2) =  8/9;    w(3) =  w(1);

  case 4
    x(1) = -0.861136311594053;     x(4) = -x(1);
    x(2) = -0.339981043584856;     x(3) = -x(2);
    w(1) =  0.347854845137454;     w(4) =  w(1);
    w(2) =  0.652145154862546;     w(3) =  w(2);

  case 5
    x(1) = -0.906179845938664;     x(5) = -x(1);
    x(2) = -0.538469310105683;     x(4) = -x(2);
    x(3) =  0;
    w(1) =  0.236926885056189;     w(5) =  w(1);
    w(2) =  0.478628670499366;     w(4) =  w(2);
    w(3) =  0.568888888888889;
  case 6
    x(1) = -0.932469514203152;     x(6) = -x(1);
    x(2) = -0.661209386466265;     x(5) = -x(2);
    x(3) = -0.238619186083197;     x(4) = -x(3);
    w(1) =  0.171324492379170;     w(6) =  w(1);
    w(2) =  0.360761573048139;     w(5) =  w(2);
    w(3) =  0.467913934572691;     w(4) =  w(3);

  case 7
    x(1) = -0.949107912342759;     x(7) = -x(1);
    x(2) = -0.741531185599394;     x(6) = -x(2);
    x(3) = -0.405845151377397;     x(5) = -x(3);
    x(4) =  0;
    w(1) =  0.129484966168870;     w(7) =  w(1);
    w(2) =  0.279705391489277;     w(6) =  w(2);
    w(3) =  0.381830050505119;     w(5) =  w(3);
    w(4) =  0.417959183673469;

  case 8
    x(1) = -0.960289856497536;     x(8) = -x(1);
```

```
    x(2) = -0.796666477413627;      x(7) = -x(2);
    x(3) = -0.525532409916329;      x(6) = -x(3);
    x(4) = -0.183434642495650;      x(5) = -x(4);
    w(1) =  0.101228536290376;      w(8) =  w(1);
    w(2) =  0.222381034453374;      w(7) =  w(2);
    w(3) =  0.313706645877887;      w(6) =  w(3);
    w(4) =  0.362683783378362;      w(5) =  w(4);

  otherwise
    error(sprintf('Gauss quadrature with %d nodes not supported',nn));

end
```

629
630

11.3.4 节点和权值的计算

在本节中，我们介绍寻找Gauss-Legendre求积法中节点和权值的两种办法。首先，介绍只适合于低阶情况的手工计算过程。然后，介绍一种更复杂的方法，它由Golub和Welch [33] 首先提出。

低阶公式的手工计算 一个Gauss-Legendre求积法的节点分布在相应Legendre多项式的零点上。如果这些零点已知，就可以通过方程（11-23）直接计算出权值。虽然这两步过程很简单，但是它却不适合高阶求积公式。下面的例子示范了2阶公式的相应计算。

例11.11 计算2阶Gauss-Legendre公式的节点和权值

本例示范如何直接计算低阶Gauss-Legendre公式的节点和权值。节点是相应的Legendre多项式的零点。2阶Legendre多项式为$\Phi_2(x) = (1/2)(3x^2-1)$，因此零点是方程$3x^2-1=0$的解，即

$$x_1 = -\frac{1}{\sqrt{3}},\qquad x_2 = \frac{1}{\sqrt{3}}$$

这里使用简单的代数方法得到节点，还可参考文献[13]中求更高阶Legendre多项式零点的方法。权值由方程（11-23）计算，其中L_i是$n-1$阶Lagrange插值多项式。对$n=2$，

$$L_1 = \frac{x-x_2}{x_1-x_2},\qquad L_2 = \frac{x-x_1}{x_2-x_1}$$

由刚刚定义的x_1和x_2，得

$$L_1 = \frac{x-1/\sqrt{3}}{-2/\sqrt{3}} = -\frac{\sqrt{3}}{2}\left(x-\frac{1}{\sqrt{3}}\right),\qquad L_2 = \frac{x+1/\sqrt{3}}{2/\sqrt{3}} = \frac{\sqrt{3}}{2}\left(x+\frac{1}{\sqrt{3}}\right)$$

且

$$c_1 = \int_{-1}^{1} L_1\,\mathrm{d}x = -\frac{\sqrt{3}}{2}\int_{-1}^{1}\left(x-\frac{1}{\sqrt{3}}\right)\mathrm{d}x = -\frac{\sqrt{3}}{2}\left(\frac{x^2}{2}-\frac{x}{\sqrt{3}}\right)\Bigg|_{-1}^{1} = 1$$

$$c_2 = \int_{-1}^{1} L_2\,\mathrm{d}x = \frac{\sqrt{3}}{2}\int_{-1}^{1}\left(x+\frac{1}{\sqrt{3}}\right)\mathrm{d}x = \frac{\sqrt{3}}{2}\left(\frac{x^2}{2}+\frac{x}{\sqrt{3}}\right)\Bigg|_{-1}^{1} = 1$$

高阶公式的自动计算 现在我们介绍求解任意阶数Gauss-Legendre公式的节点和权值的方法。计算会涉及到一个数值特征值求解问题。这需要注意具体数值细节和计算效率问题。有兴趣的读者可以参考附录A中关于代数特征值问题的介绍。我们计算特征值要依靠内置的`eig`函数。函数`eig`高效而且可靠，使下面介绍的方法便于编程。求节点和权值的程序由Golub 和

631

Welsch [33]创建，在Gautschi [26]中加以改进。这里介绍的是Schwarz [64]中所提供算法的一个简略版。

n阶Legendre多项式$\Phi_n(x)$的零点（根节点）是对称三对角矩阵

$$J=\begin{bmatrix} 0 & \beta_1 & & & & \\ \beta_1 & 0 & \beta_2 & & & \\ & \beta_2 & 0 & \beta_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \beta_{n-2} & 0 & \beta_n \\ & & & & \beta_{n-1} & 0 \end{bmatrix} \tag{11-24}$$

的特征值，其中

$$\beta_k=\frac{k}{\sqrt{4k^2-1}} \qquad k=1,2,\ldots,n-1$$

已知$n\times n$矩阵J，特征值问题就是要求解出特征值λ以及相应的特征向量υ，使

$$J\upsilon=\lambda\upsilon$$

一个$n\times n$矩阵最多有n个特征值–特征向量对。方程(11-24)中J的特征值是实数，且以符号相反的值成对出现。J的特征向量是正交的。既然n阶Gauss-Legendre公式的节点是n阶Legendre多项式的零点，那么得到J_n的特征值就得到了节点值，相应的权为

$$c_k=2(\upsilon_1^{(k)})^2, \qquad k=1,2,\ldots,n \tag{11-25}$$

其中$\upsilon^{(k)}$是对应于λ_k的规格化（$\|\upsilon\|_2=1$）特征向量，$\upsilon_1^{(k)}$是规格化特征向量的第一个元素。

下面的MATLAB语句计算了4阶Gauss-Legendre求积公式的节点和权值。首先，建立矩阵J：

[632]

```
>> n = 4;
>> beta = (1:n-1)./sqrt(4*(1:n-1).^2 - 1)
beta =
    0.5774    0.5164    0.5071
>> J = diag(beta,-1) + diag(beta,1)
J =
         0    0.5774         0         0
    0.5774         0    0.5164         0
         0    0.5164         0    0.5071
         0         0    0.5071         0
```

函数eig的两参数形式[V,D]=eig(A)将返回的特征向量放在矩阵V的列中，返回的特征值放在对角矩阵D的对角线上：

```
>> [V,D] = eig(J)
V =
    0.5710   -0.4170   -0.5710   -0.4170
    0.3363   -0.6220    0.3363    0.6220
   -0.4170   -0.5710    0.4170   -0.5710
   -0.6220   -0.3363   -0.6220    0.3363
D =
    0.3400         0         0         0
         0    0.8611         0         0
```

```
        0         0   -0.3400         0
        0         0         0   -0.8611
```

节点就是D中的对角线元素:

```
>> x = diag(D)
x =
    0.3400
    0.8611
   -0.3400
   -0.8611
```

为了使用Gauss-Legendre求积公式，节点需要按升序存储，这需要用带两参数的内置sort函数来实现，因为对D的任何变换V也要作相应变换，以保持特征值和特征向量之间的关系[⊖]。在下面的代码行中，x的元素按升存储，其排列次序存储在向量ix中，然后用向量ix对V按列重新排序:

633

```
>> [x,ix] = sort(diag(D));  %  sort order is saved in the ix vector
>> V(:,1:n) = V(:,ix)       %  shuffle columns of V according to ix
V =
   -0.4170   -0.5710    0.5710   -0.4170
    0.6220    0.3363    0.3363   -0.6220
   -0.5710    0.4170   -0.4170   -0.5710
    0.3363   -0.6220   -0.6220   -0.3363
```

权值通过将每个规格化特征向量的第一个元素平方再乘以2得到（参考方程(11-25)）。这些元素就是向量V的第一行:

```
>> V(1,:)
ans =
    0.5710   -0.4170   -0.5710   -0.4170
```

MATLAB可自动将特征向量规格化:

```
>> norm(V(:,1))
ans =
    1.0000

>> norm(V(:,2))
ans =
    1.0000
```

因此，不需要在计算权之前进行如下计算:

```
>> w = 2*V(1,:).^2
w =
    0.3479    0.6521    0.6521    0.3479
```

前面的语句序列是程序清单11-6中GLNodeWt函数的基础。此MATLAB代码是从Wilson和Turcotte [79,5.2节]的gaussint代码中摘录的。虽然GLNodeWt的计算量随着求积公式阶数的增加而急剧增加，但是一次调用GLNodeWt的执行时间是微不足道的。如果愿意的话，可以用GLNodeWt来产生节点和权值，这些值可以复制到GLtable中。使用NMM工具箱的makeGLtable函数可以自动实现此过程。

⊖ 第k个特征值保存在D(k,k)中，其对应的特征向量在V(:,k)中。

程序清单11-6 函数GLNodeWt 对任意阶数的Gauss-Legendre求积公式计算节点和权值

```
function [x,w] = GLNodeWt(n)
% GLNodeWt  Nodes and weights for Gauss-Legendre quadrature of arbitrary order
%           obtained by solving an eigenvalue problem
%
% Synopsis:  [x,w] = GLNodeWt(n)
%
% Input:     n = order of quadrature rule
%
% Output:    x = vector of nodes
%            w = vector of weights

%  Algorithm based on ideas from Golub and Welsch, and Gautschi.  For a
%  condensed presentation see H.R. Schwarz, "Numerical Analysis: A
%  Comprehensive Introduction", 1989, Wiley.  Original MATLAB
%  implementation by H.W. Wilson and L.H. Turcotte, "Advanced Mathematics
%  and Mechanics Applications Using MATLAB", 2nd ed., 1998, CRC Press

beta   = (1:n-1)./sqrt(4*(1:n-1).^2 - 1);
J      = diag(beta,-1) + diag(beta,1);    % eig(J) needs J in full storage
[V,D]  = eig(J);
[x,ix] = sort(diag(D));  %  nodes are eigenvalues, which are on diagonal of D
w      = 2*V(1,ix)'.^2;  %  V(1,ix)' is column vector of first row of sorted V
```

11.3.5 Gauss-Legendre求积法的复合公式

634

与Newton-Cotes公式一样，高斯求积公式也可以用在划分多个小段的较大区间中。但是实现时却很不同，这是由于高斯求积公式需要在每个小段内定位不等距的积分节点。

对积分

$$I = \int_a^b f(x)\,dx \tag{11-26}$$

进行数值逼近，要将区间$[a,b]$划分为N个相等宽度$H = (b-a)/N$的小段。在本节中，N表示小段的数目，n仍然作为每个小段中的节点数。小段宽度是H，不要将它与Newton-Cotes公式中的节点间距h混淆。使用等距小段非常方便，但不一定要这么做。每个小段的起始点是

$$x_i = a + (i-1)H, \quad i = 1,\ldots,N$$

对每个小段使用Gauss-Legendre求积公式逼近

635

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx \tag{11-27}$$

方程(11-26)中的积分值通过将每个小段的值相加得到：

$$I = \sum_{i=1}^{N} I_i \tag{11-28}$$

要使用Gauss-Legendre求积公式，必须将方程(11-27)中积分区间由$[x_i,x_{i+1}]$变成$[-1,1]$。如图11-13所示，这可以使用下列变量转换来实现：

$$z = \frac{x - x_{m,i}}{H/2}, \qquad x_{m,i} = \frac{1}{2}(x_i + x_{i+1}) \tag{11-29}$$

积分变量也必须作出相应的变化。由

$$x = x_{m,i} + \frac{zH}{2} \Rightarrow dx = \frac{H}{2}dz$$

方程（11-27）变成

$$I = \int_{-1}^{1} f(x)\frac{H}{2}dz = \frac{H}{2}\int_{-1}^{1} f\left(x_{m,i} + \frac{zH}{2}\right)dz \tag{11-30}$$

方程（11-27）到方程（11-30）适用于任何节点数的求积公式。在区间$[a,b]$上的Gauss-Legendre求积算法如下。

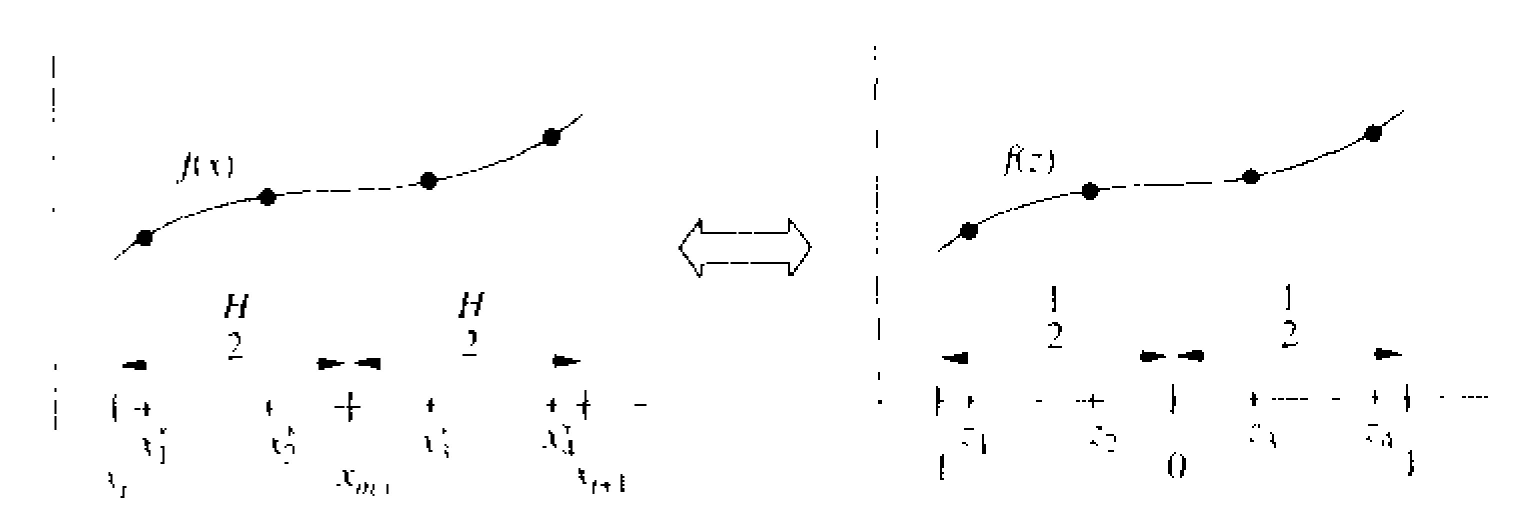


图11-13　任意小段$x_i \leqslant x \leqslant x_{i+1}$向区间$-1 \leqslant z \leqslant 1$的转换，以符合Gauss-Legendre求积公式。这里显示的是四节点求积公式

636

算法11-2　Gauss-Legendre求积法的复合公式

```
input: f(x), a, b, N, n
H=(b-a)/N
compute(or look-up) z_j, and w_j, for -1 ≤ z ≤ 1
I=0
for i=1,...,N                    (在小段上循环)
    I_i=0
    for j=1,...,n                (在小段内的节点上循环)
        x*_j=x_{m,i}+z_j H/2
        I_i = I_i+c_j f (x*_j)
    end
    I = I+I_i
end
```

符号x^*_j表示当前小段中第j个节点的x位置，上标*可以避免与第i个小段的起始边位置x_i混淆。图11-13形象地描述了x^*_j和x_i的位置。

在n节点的情况下，宽度为H的小段上复合Gauss-Legendre公式的截断误差为[11,40,70]：

$$E_{n,n}\{f\} = \frac{1}{2}C_n\left(\frac{H}{2}\right)^{2n} f^{(2n)}(\xi) = O(H^{2n}) \tag{11-31}$$

其中C_n是一个仅由n决定的因子，$f^{(2n)}$是$d^{(2n)}f/dx^{(2n)}$的简写，ξ是(a,b)中的某个点。

例11.12　用Gauss-Legendre求积法计算erf (x)

本例将要示范应用复合Gauss-Legendre求积公式来计算

$$\mathrm{erf}(1)=\frac{2}{\sqrt{\pi}}\int_0^1 e^{-x^2}\,\mathrm{d}x$$

为方便起见，令

$$I=\int_0^1 e^{-x^2}\,\mathrm{d}x$$

于是有

$$\mathrm{erf}(1)=2I/\sqrt{\pi}$$

现在，我们使用2阶复合Gauss-Legendre求积公式来计算I的值，对于一个小段，$H=1$，对于两个小段，$H=1/2$。由节点权值表或例11.11可知，2阶复合Gauss-Legendre求积公式的节点和权值为

637

$$z_1=-\frac{1}{\sqrt{3}},\quad z_2=\frac{1}{\sqrt{3}},\quad c_1=c_2=1$$

这里节点位置存在z变量中，为应用复合公式所需要进行的变换作准备。每个宽度为H的小段，I的近似值由方程(11-30)给出。

一个小段，此小段带两个节点 使用$x_j^*=z_jH/2+x_{m,i}$转换节点。对一个小段，$H=1$，故

$$x_1^*=\frac{1}{2}\left(-\frac{1}{\sqrt{3}}\right)+\frac{1}{2}=0.2113248654$$

$$x_2^*=\frac{1}{2}\left(\frac{1}{\sqrt{3}}\right)+\frac{1}{2}=0.7886751345$$

在转换后的节点处计算被积函数，得

$$f(x_1^*)=\exp[-(0.2113248654)^2]=0.9563242988$$
$$f(x_2^*)=\exp[-(0.7886751345)^2]=0.5368650777$$

应用积分公式，得

$$I=\frac{H}{2}(c_1f(x_1^*)+c_2f(x_2^*))=0.5(0.9563242988+0.5368650777)=0.7465946883$$

于是用带两个节点的单小段对erf (x)的数值逼近为

$$2I/\sqrt{\pi}=0.8424418925$$

它与erf (1)的精确值有 -0.0002589004的绝对误差，这和例11.9得到的结果相同。

两个小段，每个小段两个节点 对两个小段来说，$H=1/2$。对第一个小段有

$$x_1^*=\frac{1}{4}\left(-\frac{1}{\sqrt{3}}\right)+\frac{1}{4}=0.1056624327$$

$$x_2^*=\frac{1}{4}\left(\frac{1}{\sqrt{3}}\right)+\frac{1}{4}=0.3943375673$$

638

$$f(x_1^*)=\exp[-(0.1056624327)^2]=0.9888975426$$
$$f(x_2^*)=\exp[-(0.3943375673)^2]=0.8559852648$$

故

$$I_1 = \frac{H}{2}(c_1 f(x_1^*) + c_2 f(x_2^*)) = 0.25(0.9888975426 + 0.8559852648) = 0.4612207109$$

对第二个小段有

$$x_1^* = \frac{1}{4}\left(-\frac{1}{\sqrt{3}}\right) + \frac{3}{4} = 0.6056624327$$

$$x_2^* = \frac{1}{4}\left(\frac{1}{\sqrt{3}}\right) + \frac{3}{4} = 0.8943375673$$

$$f(x_1^*) = \exp[-(0.6056624327)^2] = 0.6929295237$$

$$f(x_2^*) = \exp[-(0.8943375673)^2] = 0.4494010044$$

故

$$I_2 = \frac{H}{2}(c_1 f(x_1^*) + c_2 f(x_2^*)) = 0.25(0.6929295237 + 0.4494010044) = 0.2855826320$$

使用两个各带两个节点的小段得到的对erf(x)的数值逼近为

$$2I/\sqrt{\pi} = \frac{2}{\sqrt{\pi}}(I_1 + I_2) = 0.7468033339$$

与erf(1)的精确值有-0.0000234691的绝对误差。

MATLAB实现 程序清单11-7中的gaussQuad函数实现了算法11-2。节点的位置和权值可由程序清单11-5中的GLtable函数或程序清单11-6中的GLNodeWt函数（$n>8$）求得。对于一个求节点和权值的程序，函数gaussQuad并不比函数trapezoid或simpson明显地复杂多少。

例11.13 使用Gauss-Legendre求积法解 $\int_0^5 xe^{-x}\,dx$

程序清单11-8中的demoGauss函数通过计算 $\int_0^5 xe^{-x}\,dx$ 对Gauss-Legendre求积法进行了两项检验。第一个检验表明，小段数固定，误差是如何随着每个小段上节点数的增加而减小的。第二个检验使每个小段上的节点数固定，观察误差如何随小段数的增加而减小。第二个检验允许计算截断误差表达式中H的阶数（另见例11.6和例11.8）。可选的输入参数用来指定第一个检验中使用的小段数和第二个检验中每个小段使用的节点数。 639

程序清单11-7 函数gaussQuad实现了对任意被积函数和积分区间的复合Gauss-Legendre求积法

```
function I = gaussQuad(fun,a,b,npanel,nnode,varargin)
% gaussQuad  Composite Gauss-Legendre quadrature
%
% Synopsis:  I = gaussQuad(fun,a,b,npanel)
%            I = gaussQuad(fun,a,b,npanel,nnode)
%            I = gaussQuad(fun,a,b,npanel,nnode,arg1,arg2,...)
%
% Input:  fun    = (string) name of m-file that evaluates f(x)
%         a,b    = lower and upper limits of the integral
%         npanel = number of panels in interval [a,b]
%         nnode  = (optional) number of nodes on each subinterval
%                  Default:  nnodes=4
%         arg1,arg2,... = (optional) parameters passed through to fun
```

```
%
% Output: I = approximate value of the integral from a to b of f(x)*dx

if nargin<5,  nnode = 4;  end

if nnode<=8
  [z,wt] = GLTable(nnode);           % Look up nodes and weights,
else                                 % or if n is too big for the table,
  [z,wt] = GLNodeWt(nnode);          % compute the nodes and weights
end
H = (b-a)/npanel;                    % Size of each panel
H2 = H/2;                            % Avoids repeated computation of H/2
x = a:H:b;                           % Divide the interval

I = 0;                               % Initialize sum
for i=1:npanel
  xstar = 0.5*(x(i)+x(i+1)) + H2*z;  % Evaluate 'fun' at these points
  f = feval(fun,xstar,varargin{:});
  I = I + sum(wt.*f);                % Add contribution of this subinterval
end
I = I*H2;                            % Factor of H/2 for each subinterval
```

[640]

程序清单11-8 函数demoGauss显示了Gauss-Legendre求积法的截断误差如何随着小段数和小段内节点数的增加而减小

```
function demoGauss(np,nn)
% demoGauss  Use Gauss-Legendre quadrature to integrate x*exp(-x) on [0,5]
%
% Synopsis:  demoGauss
%            demoGauss(np)
%            demoGauss(np,nn)
%
% Input:    np = (optional) panels used in node refinement test.  Default: np=1
%           nn = (optional) nodes used in panel refinement test.  Default: nn=3
%
% Output:   Tables integral values obtained with Gauss-Legendre quadrature
%           as function of increasing nodes and increasing number of panels

if nargin<1;  np = 1;  end
if nargin<2;  nn = 3;  end
a = 0;  b = 5;   Iexact = -exp(-b)*(1+b) + exp(-a)*(1+a);

% --- Truncation error as function of number of nodes
H = (b-a)/np;
fprintf('\nGauss-Legendre quadrature with %d panels, H = %f\n',np,H);
fprintf('\n nodes        I            error\n');
for n = 1:8
  I = gaussQuad('xemx',a,b,np,n);
  fprintf('%4d  %14.10f %12.2e\n',n,I,I - Iexact)
end

% --- Truncation error as function of panel size
fprintf('\nGauss-Legendre quadrature with %d nodes\n',nn);
fprintf('\n panels    H             I            error       alpha\n');
k = 1;   %  alpha is computed only if k>1
```

```
for npanel = 1:8
  I = gaussQuad('xemx',a,b,npanel,nn);
  err = I - Iexact;
  H = (b-a)/npanel;                % Compute H for output only
  fprintf(' %4d %10.5f %14.10f %12.2e',npanel,H,I,err);

  if k>1,  fprintf('  %8.2f\n',log(err/errold)/log(H/HHold)),
  else,    fprintf('\n');  end
  HHold = H;  errold = err;   k = k + 1;  %  prep for next stepsize
end
```

641

在缺省参数下运行demoGauss，得

```
>> demoGauss

Gauss-Legendre quadrature with 1 panels, H = 5.000000

 nodes        I              error
    1    1.0260624828     6.65e-02
    2    1.1093615949     1.50e 01
    3    0.9744587198     1.49e-02
    4    0.9601822632     6 10e-04
    5    0.9595862620     1.39e-05
    6    0 9595725226     2.05e-07
    7    0.9595723201     2.10e-09
    8    0.9595723180     1.58e-11

Gauss -Legendre quadrature with 3 nodes

 panels     H            I              error       alpha
    1    5.00000    0.9744587198     1.49e-02
    2    2.50000    0.9600305785     4.58e-04       5.02
    3    1.66667    0.9596190800     4.68e-05       5.63
    4    1.25000    0.9595811126     8.79e-06       5.81
    5    1.00000    0.9595746842     2.37e-06       5.88
    6    0.83333    0.9595731218     8.04e-07       5.92
    7    0.71429    0.9595726395     3.22e-07       5.94
    8    0.62500    0.9595724631     1.45e-07       5.96
```

第二个检验得到$\alpha \simeq 6$，这与$n = 3$时方程（11-31）对截断误差的估计一致。一般地，在函数求值数目（即每个小段的节点数）相同的条件下，Gauss-Legendre公式产生的截断误差比梯形公式或Simpson公式产生的要小得多。下例就对这些方法进行量化比较。

例11.14 积分公式的计算效率比较

本例中，使用复合的梯形、Simpson和Gauss-Legendre公式的截断误差公式来估计这些积分方法的相对计算效率。令n_f为被积函数的求值数目，N为复合积分公式的小段数。对每种方法，复合公式中的小段大小为

$$H = \frac{b-a}{N}$$

梯形公式、Simpson公式和Gauss-Legendre公式中所有小段的节点数(n_f)分别为：$n_f = N+1$、$n_f = 2N+1$和$n_f = Nn$。

梯形公式的截断误差为$O(h^2)$，其中$h=H$为节点间的距离。对Simpson公式，$h = H/2$，代入

[642] $H = (b-a)/N$和$N = n_l - 1$，只保留最高次项，梯形公式的截断误差变为$O(n_l^{-2})$。对其他积分方法进行相同操作可得下表结果：

方法	截断误差 对h的依赖	对n_l的依赖
梯形	$O(h^2)$	$O(n_l^{-2})$
Simpson	$O(h^4)$	$O(n_l^{-4})$
Gauss-Legendre	$O(H^{2n})$	$O(n_l^{-2n})$

这些表达式足以表明，$n>2$时，Gauss-Legendre公式是计算上最有效的算法，在n增大时它甚至会更有效。由此可推断，增加Gauss-Legendre的阶数比增加其区间内的小段数更有意义。但是，这并非总是正确的，因为上表中的表达式并不包括方程(11-31)中的$f^{2n}(\xi)$项。一般情况下，最好将积分区间分为几个小段然后增加求积公式的阶数。

将已知积分的截断误差作为n_l的函数，测量它可以对计算效率进行量化比较。函数compIntRules（在NMM工具箱中，此处未列出）可得到用梯形、Simpson和四节点Gauss-Legendre公式计算$\int_0^5 xe^{-x}\,dx$时产生的截断误差。运行compIntRules得

```
>> compIntRules

Truncation error versus number of function evaluations

   Trapezoid              Simpson               GL 4 node
  nf      error         nf      error         nf      error
   2    -8.75e-01        3    -2.47e-01        2    1.50e-01
   4    -2.09e-01        5    -7.07e-03        4    6.10e-04
   8    -4.26e-02        9    -2.66e-04        8    4.59e-06
  16    -9.46e-03       17    -1.28e-05       16    2.18e-08
  32    -2.22e-03       33    -7.07e-07       32    8.99e-11
  64    -5.39e-04       65    -4.15e-08       64    3.56e-13
 128    -1.33e-04      129    -2.51e-09      128    1.22e-15
 256    -3.29e-05      257    -1.55e-10      256    0.00e+00
```

每种方法下面的两列数据分别是函数求值数目和截断误差，图11-14中也表示出相同的信

[643]

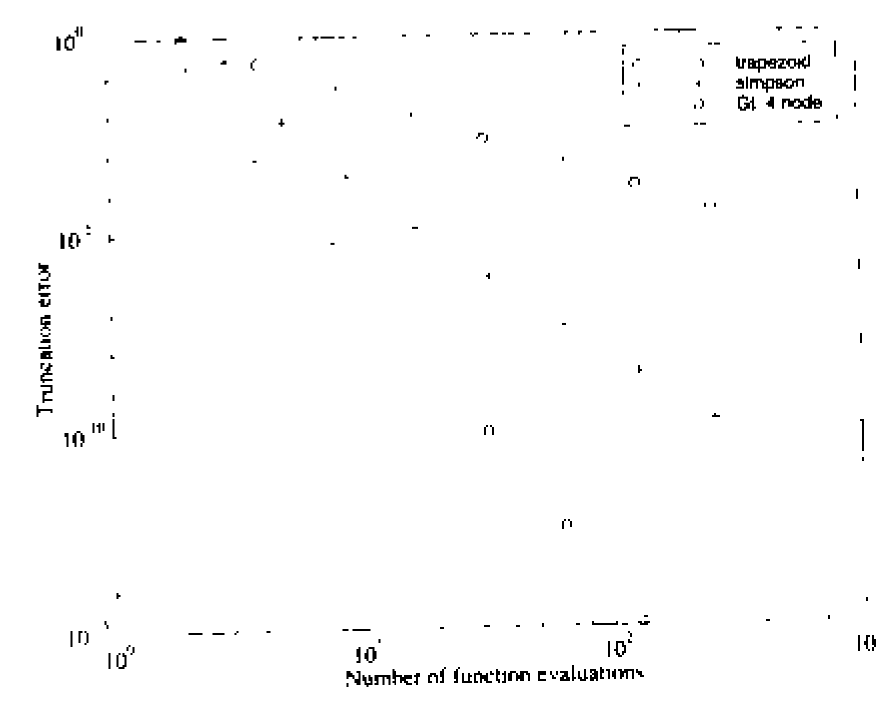


图11-14　在使用梯形公式、Simpson公式和Gauss-Legendre公式计算$\int_0^5 xe^{-x}\,dx$时，截断误差作为函数求值个数的函数

息。表格结果和图11-14中的图示确认了这三种方法的理论截断误差估计的比较。四节点Gauss-Legendre公式明显地比其他两种公式有效。付出相同的代价——对被积函数的求值个数相同，四节点Gauss-Legendre公式所得积分值的估计截断误差要小得多。另外，随着被积函数求值个数的增加，四节点Gauss-Legendre公式的截断误差减小速度更快。

如果前面的计算使用二节点和八节点的Gauss-Legendre公式，图11-14会如何变化（参见习题20）？

11.4 自适应求积法

自适应求积法尝试自动和最优化地进行积分的数值计算。其中，自动计算免去了由用户来划分积分区间小段数的任务。自适应求积方法从初始的小段分布开始，如果积分的截断误差超过用户自定义的容差就自动增加小段的数目。最优化只有在允许当积分区间上改变小段大小时才会出现。这样，就只在需要时才使用较小的小段，它需要对被积函数求更多的值。 [644]

如果一切进展良好的话，自适应求积法的结果就是一个落在用户自定义容差内的积分值。另外，相对于将整个区域划分为同样宽度的小段，它得到数值结果所运行的时间会更短。然而，不应该把自适应求积法看成是完全自动的。虽然容差的使用免去了确定小段数的任务，但是用户应该确定容差对待处理的问题是否合适。一个很好的办法是通过使用较小（可能是较大）的容差反复计算来确定积分值。稍作合理的努力，自适应方法确实提供了一个很有用的解决数值积分问题的工具。

本节使用小段长度的局部细分（local refinement）法推导自适应算法。另一种不同于自适应算法的方法是基于小段大小的对称等距细分法。使用等距细分的一个通用方法是Romberg积分。Romberg积分对平滑的被积函数非常有效（也就是说，在积分区间的有限个子区间内对小段细分并没有太大的优势）。文献[10、64、70]中有关于Romberg积分理论的推导，Press等人在[61]中给出了Romberg积分的C程序，Mathews和Fink [53]及Fausett[22]提供了它的MATLAB实现。

QUADPACK[59]是自适应求积法的一个Fortran库。QUADPACK中的自适应方法包括了内置的quad和quad8函数，比此处提供的程序要更完善精练。QUADPACK程序（大多数）的一个显著优点就是它们的子区间采用全局自适应（global adaptation）进行细分，子区间应用基本公式。下面介绍的自适应方法就只使用了局部自适应（local adaptation）。

局部自适应方法对给定子区间连续细分，直到最好的子区间细分上的积分值符合定义容差的规定，或者达到规定的递归界限。对一些被积函数来说，在某个子区间上的积分值接近零，那么即使积分结果很精确，局部细分也会达到递归的界限。局部自适应方法在局部子区间细分的过程中无法知道积分的真实值正在接近零，这种情况下，局部误差对全局误差的影响就可以忽略不计。

全局自适应方法将所有子区间上的估计误差相加，以此估计整个积分的误差。全局方法的程序逻辑更复杂些，但优点是所有的计算都直接朝着对全局误差有影响的x的范围努力，这样就避免了那些对全局积分影响很小的局部子区间可能造成的混乱。 [645]

11.4.1 基于Simpson公式的自适应积分

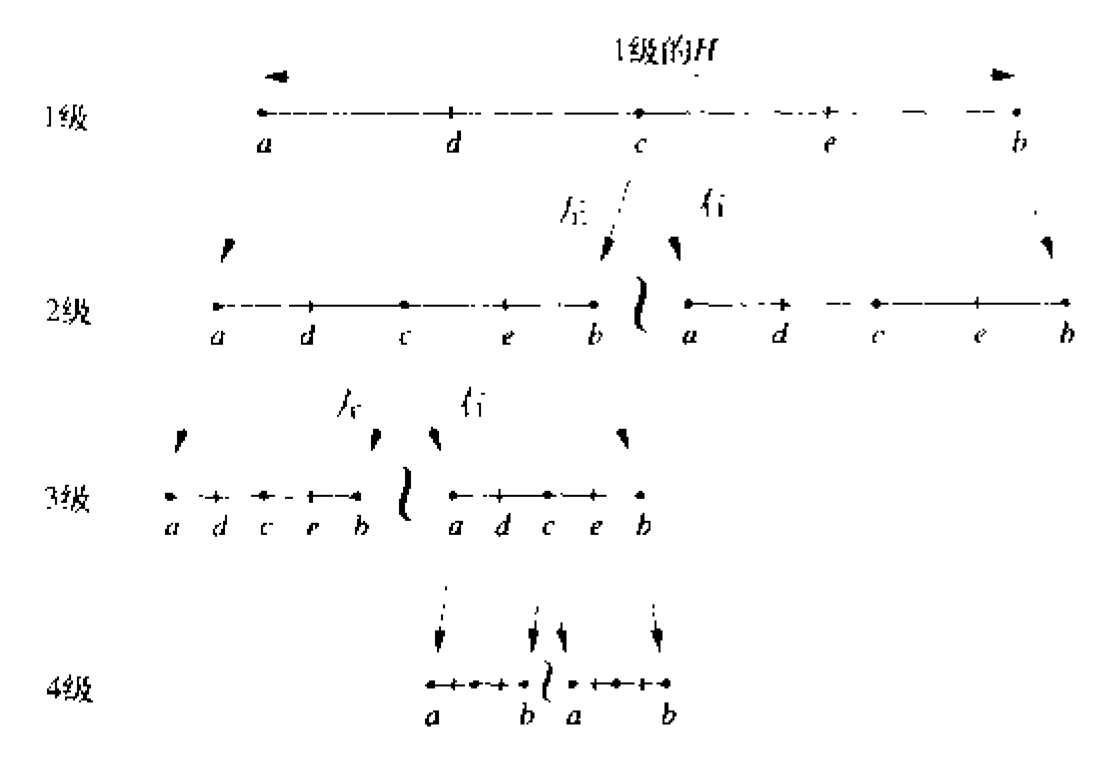
本节开发了基于Simpson公式的自适应数值积分程序。所得到的m文件函数与内置的quad

函数相似[Θ]。算法理论和程序逻辑都是基于Cheney和Kincaid[10]中的介绍，另外Coute和de Boor[11]也给出了相似的推导。

自适应算法 对积分

$$I=\int_a^b f(x)\,dx \tag{11-32}$$

的数值逼近可通过对子区间应用Simpson基本公式来计算。每个子区间再独立地划分为更小的子区间。图11-15表示了对自适应Simpson公式的4级细分序列。计算一开始对1级的情况应用Simpson公式。计算出1级区间的中点$c=(a+b)/2$，然后再对每个半区间$[a,c]$和$[c,b]$应用Simpson公式。将大小为$H=b-a$的区间上的截断误差与两个大小为$H/2$的半区间上的截断误差的和相比较。如果二者的差比用户自定义的容差大，就将两个半区间分别作为起点再进行下一级细分。



646

图11-15 自适应Simpson求积方法的4级细分

图11-15中用从1级到2级的左右箭头来描述细分步骤。在2级中，左段b点的x值与右段a点的x值相等。当细分开始后，左右段的细分就分别进行，故而我们可以重新使用符号a、b、c、d和e。

是否对一个区间细分依赖于对截断误差的估计。Simpson基本公式为

$$I=\int_a^b f(x)\,dx=\frac{h}{3}(f(a)+4f(c)+f(b))-\frac{h^5}{90}f^{(4)}(\xi),\qquad h=\frac{b-a}{2} \tag{11-33}$$

其中点$\xi(a<\xi<b)$是未知的，但只要a和b确定，它就是常量。注意$h=H/2$是定义Simpson公式的二个节点之间的距离，$H=b-a$是应用Simpson基本公式的小段的长度。

方程(11-33)可变化为

$$I_1=S_1(h)+E_1(h) \tag{11-34}$$

其中下标1表示此处的积分值是1级细分情况下求得的。S_1是1级上的积分数值逼近：

$$S_1(h)=\frac{h}{3}(f(a)+4f(c)+f(b)) \tag{11-35}$$

Θ MATLAB第6版虽然在quad中同样使用了Simpson公式作为积分方法的核心，但是却使用了更完善的算法。

E_1是对应于S_1的截断误差，

$$E(h) = -\frac{h^5}{90} f^{(4)}(\xi) \tag{11-36}$$

由于包含了截断误差，故而方程（11-34）是精确的。

方程（11-32）中的积分也可以由如下算法所示的两个2级上的积分相加来精确求值

$$I_2 = I_\ell + I_r = \int_a^c f(x)\,\mathrm{d}x + \int_c^b f(x)\,\mathrm{d}x \tag{11-37}$$

其中下标ℓ和r分别表示左右子区间。对两个半区间分别应用Simpson公式，得

[647]

$$I_\ell = \int_a^c f(x)\,\mathrm{d}x = S_\ell(h/2) + E_\ell(h/2)$$

和

$$I_r = \int_c^b f(x)\,\mathrm{d}x = S_r(h/2) + E_r(h/2)$$

其中

$$S_\ell(h/2) = \frac{h}{6}(f(a)+4f(d)+f(c)), \qquad E_\ell(h/2) = -\frac{(h/2)^5}{90} f^{(4)}(\xi_\ell)$$

$$S_r(h/2) = \frac{h}{6}(f(c)+4f(e)+f(b)), \qquad E_r(h/2) = -\frac{(h/2)^5}{90} f^{(4)}(\xi_r)$$

注意截断误差项中的导数在它们的各个区间内分别计算：$a<\xi_\ell<c$，$c<\xi_r<b$

定义2级的积分数值及其截断误差S_2和E_2如下：

$$S_2 = S_\ell(h/2) + S_r(h/2) \text{ 和 } E_2 = E_\ell(h/2) + E_r(h/2)$$

于是，方程（11-37）可以重写为

$$I_2 = S_2 + E_2 \tag{11-38}$$

I_1和I_2精确地相等，但是，一般情况下$S_1 \neq S_2$，这是因为1级和2级的截断误差不同。S_1和S_2由数值方法求得。但是，在一般情况下ξ、ξ_ℓ和ξ_r是未知的，所以E_1和E_2只能通过估计得出。自适应算法会一直细分下去，直到$|S_1 - S_2|$比用户自定义的容差小。为推导此收敛判别式我们要估计E_1和E_2的相对值。

2级截断误差为

$$E_1 = -\frac{h^5}{90} f^{(4)}(\xi) \qquad E_2 = -\frac{(h/2)^5}{90} f^{(4)}(\xi_\ell) - \frac{(h/2)^5}{90} f^{(4)}(\xi_r)$$

假设

$$f^{(4)}(\xi) = f^{(4)}(\xi_\ell) = f^{(4)}(\xi_r) = C \tag{11-39}$$

其中C为未知常数，于是

$$E_2 = -\frac{2h^5}{2^5\,90}C = -\frac{1}{16}\frac{h^5}{90}C = \frac{E_1}{16} \tag{11-40}$$

上式可以作出估计，当由1级转到2级下计算积分时，所期望的改进是多少。

方程（11-40）也可以用来估计数值积分对真实值的接近程度。令方程（11-34）的右边和方程（11-38）的右边相等，整理得

[648]

$$S_1 - S_2 = E_2 - E_1$$

代入方程（11-40）解出E_2来，得

$$E_2 = \frac{1}{15}(S_2 - S_1)$$

将此结果代入方程（11-38），整理得

$$I_2 - S_2 = \frac{1}{15}(S_2 - S_1) \tag{11-41}$$

假设在$[a,b]$上$f^{(4)}(x) = C$，此方程就是对积分误差的估计。实践中，用户会在可接受的积分误差上提供一个绝对容差δ。这样，如果

$$\frac{1}{15}|S_2 - S_1| < \delta \tag{11-42}$$

成立，则S_2的值就可以作为局部积分值。否则，就递归地对2级进一步细分（也就是说，对2级的每个子区间像1级的初始区间那样处理）。

前面的推导可能对任意细分级都适用，主要组成是在子区间上计算积分的公式——方程（11-35），以及估计积分值误差的公式——方程（11-41）。

计算方程（11-32）的2级递归自适应求积算法如下。

算法11-3　用2级自适应的Simpson公式求积

I = adaptSimp(f, a, b, δ)
input: f、a、b、δ　　　　（f是计算$f(x)$的程序名称）
S_1 = simpson(f, a, b, $(b-a)/2$)　　　　（1级Simpson公式）
S_2 = simpson(f, a, b, $(b-a)/4$)　　　　（2级Simpson公式）
if $(1/15)|S_1 - S_2| < \delta$, accept $I = S_2$
otherwise, refine:
　$c = (a+b)/2$
　I_{left} = adaptSimp(f, a, c, $\delta/2$)
　I_{right} = adaptSimp(f, c, b, $\delta/2$)
　$I = I_{left} + I_{right}$
return

649

注意，如果需要将区间细分，输入容差δ要除以2。当区间细分时，就用两个半区间的积分和来代替整个区间的积分，容差除以2，这样在半区间上的误差和就不会超过初始容差。

实现　MATLAB支持函数的递归调用，所以实现自适应算法的程序逻辑并不难。递归函数会自我调用，且一般用新的输入数据来处理初始问题的子集。当I_{left}和I_{right}需要调用adaptSimp来计算时，算法11-3就要用到递归。对递归函数不熟悉的读者可以研究一下NMM工具箱中的`recursiveIndent`函数。

递归函数会自动进行小段的多级连续细分。我们只需要定义两个核心操作：计算小段的粗略和精细积分值并决定是否接受粗略积分值。为防止无限循环，函数必须有一个内部的停止判断准则来结束循环，从而在函数向自己返回部分积分值时对递归的展开进行初始化。

程序清单11-9中的`adaptSimpson`函数实现了算法11-3。此函数的调用方法为：

```
I = adaptSimpson(fun,a,b)
I = adaptSimpson(fun,a,b,tol)
I = adaptSimpson(fun,a,b,tol,maxLevel)
```

其中*fun*是一个包含m文件的名字的字符串，该m文件由用户自定义，用来计算被积函数$f(x)$。积分的上下限分别为a和b。可选参数tol是应用于方程（11-42）的绝对容差δ。可选参数maxLevel决定子区间$[a,b]$上递归划分级别数的界限。子区间的数目没有限制，只限制局部细分的“深度”。maxLevel的缺省值为10，表示当遇到的子区间大小为

$$\frac{b-a}{2^{n}} \approx 0.001(b-a)$$

时，adaptSimpson函数就会发出警告。在递归中止之前，还可以进行2次额外细分。

为将算法11-3转换成MATLAB语句，需要采用一些中间步骤来改进算法性能。其中，S_1和S_2的值可以用如下语句计算：

```
h2 = (b-a)/4;
f = feval(fun,a:h2:b);
s1 = (f(1) + 4*f(3) + f(5))*2*h2/3;
s2 = (f(1) + 4*f(2) + 2*f(3) + 4*f(4) + f(5))*h2/3;
```

650

程序清单11-9 函数adaptSimpson用函数的递归调用来实现基于Simpson公式的自适应求积法

```
function I = adaptSimpson(fun,a,b,tol,maxLevel,level)
% adaptSimpson  Adaptive numerical integration based on Simpson's rule
%
% Synopsis:  I = adaptSimpson(fun,a,b)
%            I = adaptSimpson(fun,a,b,tol)
%            I = adaptSimpson(fun,a,b,tol,maxLevel)
%
% Input:  fun   = (string) name of m-file that evaluates f(x)
%         a,b   = lower and upper limits of the integral
%         tol   = (optional) absolute tolerance on error. Default:  tol = 5e-6
%         maxLevel = (optional) limit on number of refinements.
%                    Warnings are issued when maxLevel is reached.
%                    Refinement is halted when maxLevel+2 consecutive
%                    refinements are attempted. Default:  maxLevel = 10
%
% Output:    I = approximate value of the integral of f(x)*dx from a to b

if nargin<4,  tol = 5e-6;     end
if nargin<5,  maxLevel = 10;  end
if nargin<6,  level = 0;      end     %  level~=0 only on recursive call

level = level + 1;                    %  current refinement level
h2 = (b-a)/4;                         %  Node spacing on (h/2) level
f = feval(fun,a:h2:b);                %  Evaluate all f(x) for this level
s1 = (f(1) + 4*f(3) + f(5))*2*h2/3;                   %  h   level
s2 = (f(1) + 4*f(2) + 2*f(3) + 4*f(4) + f(5))*h2/3;   %  h/2 level

% --- Prevent infinite loop if recursion limit is exceeded by more than 2
if level>maxLevel
  warning(sprintf('In adaptSimpson: level = %d exceeds limit\n',level));
  if level>maxLevel+2
    fprintf('Refinement halted, recursion limit exceded by factor of 2\n');
    I = s2;  return;   %  This "return" will stop the recursion
  end
end
```

```
% --- Refine more, or accept?
if abs(s2-s1) < 15*tol        % Is difference between levels significant?
  I = s2;  return;            % Yes: accept fine level result, stop recursion
else                          % No:  subdivide again.
  c = a + (b-a)/2;
  I =   adaptSimpson(fun,a,c,tol/2,maxLevel,level) ...   % I_left
      + adaptSimpson(fun,c,b,tol/2,maxLevel,level);      % I_right
end
```

[651]

语句：=feval(fun,a:h2:b)中，S_1和S_2都在所有5个点上计算$f(x)$。这需要对m文件fun进行向量化，即输入一个x值向量，fun函数要返回一个$f(x)$值向量。

例11.15 humps的自适应积分

内置humps函数用来计算如下方程：

$$f(x)=\frac{1}{(x-0.3)^2+0.01}+\frac{1}{(x-0.9)^2+0.04}-6 \tag{11-43}$$

如图11-16的上半部分所示，$f(x)$在$x=0.3$和$x=0.9$两处有局部极大值。此函数在一个很窄的x值范围上同时包含了平缓和陡峭的区域，很是有趣，非常适合用来检验自适应求积法。

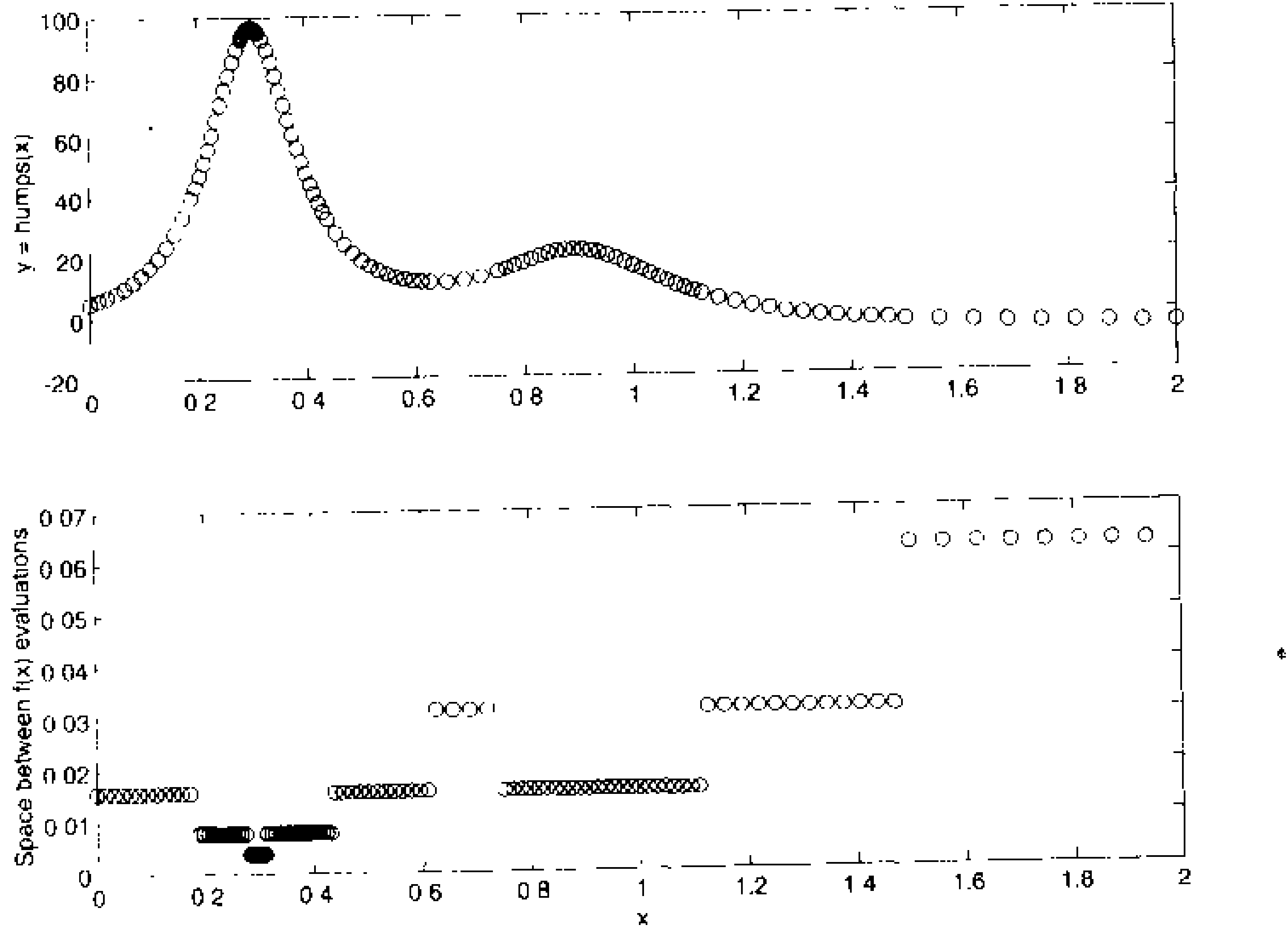


图11-16 一维humps函数图以及它用demoSimpson积分时相邻$f(x)$值的输出距离图。这些图由demoAdaptSimp(5e-4)产生，此函数使用了adaptSimpson函数的一个修改版本，称为adaptSimpsonTrace函数

[652]

程序清单11-10中的demoAdaptSimp函数将自适应Simpson公式积分用于humps函数的计算，可以很容易得出积分的精确值（比较练习3）。函数demoAdaptSimp中使用了adaptSimpson函数的一个修改版本，被称为adaptSimpsonTrace（见NMM工具箱中的

程序清单11-10 函数demoAdaptSimp用自适应Simpson公式对humps函数积分

```
function demoAdaptSimp(tol)
% demoAdaptSimp  Integrate humps(x) with adaptive Simpson's rule
%
% Synopsis:  demoAdaptSimp
%            demoAdaptSimp(tol)
%
% Input:   tol = (optional) absolute tolerance on truncation error in
%                evaluation of the integral.  Default:  tol = 5e-3
%
% Output:  Value of numerical approximation to the integral and the error;
%          Min and max spacing of points along x used in evaluating integrand
if nargin<1,  tol=5e-3;  end
a = 0;  b = 2;  Iexact = humpInt(a,b);

flops(0);   [s,x] = adaptSimpsonTrace('humps',a,b,tol);  f = flops;
dx = diff(x);
fprintf('\n\tMinimum and maximum spacing     = %g    %g\n',min(dx),max(dx));
fprintf('\tExact value of the integral     = %g\n',Iexact);
fprintf('\tNumerical value of the integral = %g\n',s);
fprintf('\tError (I - Iexact)              = %g\n',s-Iexact);
fprintf('\tFlops                           = %g\n',flops);

subplot(2,1,1);   plot(x,humps(x),'o');    ylabel('y = humps(x)');
subplot(2,1,2);   plot(x(1:end-1),abs(diff(x)),'o');
xlabel('x');      ylabel('Space between f(x) evaluations');
```

integrate目录)。函数adaptSimpsonTrace可以更容易地用来研究自适应Simpson公式的特性，它产生了包含被积函数求值所在的所有x值的向量。除非用于诊断，否则不必要列出x值，所以demoAdaptSimp函数中就没有包含此项功能。demoAdaptSimp接受返回的x值，并报告出$f(x)$相邻求值间的最小和最大间距。在缺省的容差参数下，运行demoAdaptSimp函数可得图11-16和如下输出结果：

```
>> demoAdaptSimp

Minimum and maximum spacing     = 0.0078125    0.125
Exact value of the integral     = 29.3262
Numerical Value of the integral = 29.3159
Error (I - Iexact)              = -0.0102954
Flops                           = 1875
```

图11-16的上半部分描述了自适应求积法中计算方程(11-43)的区域，下半部分描述了相邻$f(x)$求值的间距，正如期望的那样，相邻$f(x)$求值间距最小的区域就是$f(x)$变化最剧烈的地方（对比上下图）。

由函数demoAdaptSimp的打印输出可以看出，绝对误差的大小超过了缺省容差（5×10^{-3}）的两倍。虽然这很令人烦恼，但是adaptSimp区间细分的停止准则取决于对截断误差而不是真实误差的估计，所以这个结果也在意料之中。下面可以看到，这种对截断误差的严重偏离只在tol值相对较大时才发生。

在容差范围内运行demoAdaptSimp可得如下结果：

tol	min(dx)	max(dx)	error	flops
5×10^{-2}	0.0156	0.125	-2.8×10^{-2}	1125
5×10^{-3}	0.00781	0.125	-1.0×10^{-2}	1875
5×10^{-4}	0.00391	0.0625	-1.5×10^{-5}	3975
5×10^{-5}	0.00195	0.0625	-7.0×10^{-6}	6825
5×10^{-6}	0.00195	0.0313	-7.4×10^{-7}	12675

当容差减小时，绝对误差也降低了。注意，只有在tol= 5×10^{-3}时绝对误差超过了容差。正如所料想的那样，求积分的工作量（由浮点操作次数度量）随着容差的降低而增加了，这是因为需要对被积函数进行更多的计算来改进对积分的估计。

使用自适应求积法来对积分值未知的被积函数求积时，最好使用一系列更小的、更紧密的收敛容差。当容差减小时，积分数值应该会接近一个常数，这表明了在求解中对精度细分是成功的。

其他的自适应积分算法使用更复杂的判定标准来中止局部细分，从而增大了满足用户自定义容差的可能性[59、25]。

11.4.2 内置quad和quad8函数

在MATLAB 5和更早的版本中，内置函数quad和quad8使用自适应数值求积法来求指定容差内的积分值。函数quad类似于上节讨论的adaptSimpson函数，两者间的一个重要差别就是quad不用截断误差的直接估计值，而是将子区间上的积分一直细分，直到相邻两个细分级的积分差小于容差值。函数quad8使用与quad相同的逻辑，但它不使用Simpson公式，而是使用九节点Newton-Cotes公式[⊖]（见方程(11-14)）。同样求得期望容差内的积分，函数quad8有希望比quad和adaptSimpson对被积函数作更少的计算。

654

函数quad和quad8的语法相同，任一个都可以使用下列方法调用（将其中的quad换成quad8就是quad8的调用语法）：

```
I = quad(fun,a,b)
I = quad(fun,a,b,tol)
I = quad(fun,a,b,tol,trace)
I = quad(fun,a,b,tol,trace,arg1,arg2, ... )
```

最少的输入参数是*fun*、*a*和*b*，其中*fun*是字符串，它包含了求$f(x)$值的m文件的名称，*a*和*b*是积分的上限和下限。可选的*tol*参数是一个标量或向量，其中标量值给出积分值的相对容差，而向量值则同时描述了误差的相对容差和绝对容差，即

```
tol = [rel_tol abs_tol]
```

可选参数*trace*的值只要非零，就会产生被积函数图形。此图与图11-16中的上图相似，都表示了自适应算法计算$f(x)$的点的位置。*arg*1, *arg*2...的值不在quad或quad8中直接使用，而是传给由参数*fun*指定的用户自定义m文件。

一旦函数$f(x)$写成m文件代码或内嵌函数，quad和quad8都可以使用并能比较结果。另外，应该检查积分值对*tol*参数的敏感性。如果积分值随着*tol*值的改变而改变，相对和绝对容差就应该有步骤地减小，直到积分值接近一个极限。

655

⊖ quad8的序言说明它是一个使用“8个小段的自适应递归Newton-Cotes公式”的函数。这里的“小段”意为节点间的距离，（即当基本公式在9个点上计算时，有$h=(b-a)/8$）。但是本章中“小段”用来描述求积方法的基本公式的子区间宽度，$H=(b-a)/N$。

例11.16 内置函数quad示范

本例中，使用quad和quad8来对例11.15中的函数humps积分。在缺省的容差参数下，有

```
>> format long                  % show lots of digits
>> quad('humps',0,2)
ans =
  29.32622038887953

>> quad8('humps',0,2)
ans =
  29.32621734123175
```

取6位有效数字，两种方法的结果相等。积分的精确值是

```
>> format long
>> humpInt(0,2)
ans =
     29.32621380439114
```

quad和quad8都给出了在6位有效数字上的正确结果。

使用更小的收敛容差重复以上计算，会分别改变由quad和quad8返回的第7位和第8位有效数字：

```
>> quad('humps',0,2,1e-4)
ans =
  29.32621406442962

>> quad8('humps',0,2,1e-4)
ans =
  29.32621386300150
```

在这个更小的容差上，quad8增加了3位正确有效数字，quad增加了1位正确有效数字。

从这些例子可以看出，函数quad和quad8返回的积分估计值几乎相同。但是也存在一些被积函数，使用这两个函数对其积分时可能有一个或两个都不能满足用户自定义的容差，这样，对两种方法的结果进行比较就成了积分求值中一个非常重要的步骤。

对良性(well-behaved)的积分来说，函数quad和quad8的不同并不在于计算积分值的精度差异，而在于获得指定容差内积分所花费的工作量大小。一般地，quad8会较少调用计算被积函数的程序。程序清单11-11中的deomQuad函数通过对相关容差参数有步骤地细分来求humps函数的数值积分，它分别给出了积分的精确值与quad和quad8返回值的差。使用内置的flops函数来度量求积分过程中的计算量。运行demoQuad得 656

```
>> demoQuad
 -----  quad -----      ----- quad8 -----
    tol          error      flops       error        flops
  5.00e-02     2.91e-02     914       6.56e-01      608
  5.00e-03     2.54e-05    2552       5.06e-05     1577
  5.00e-04     3.30e-06    5009       3.54e-06     2223
  5.00e-05     1.94e-07   10924       1.60e-08     3192
```

图11-17对这些结果数据做了图形总结。减小tol的值也减小了积分数值中的绝对误差。记住，tol是一个用来控制细分的参数。一般地，并不能保证积分的相对或绝对误差满足tol值所隐含的误差标准，因为quad或quad8不可能知道积分的真实值。当然，这也就是使用quad和quad8以及其他各种数值积分算法的一个真正理由。

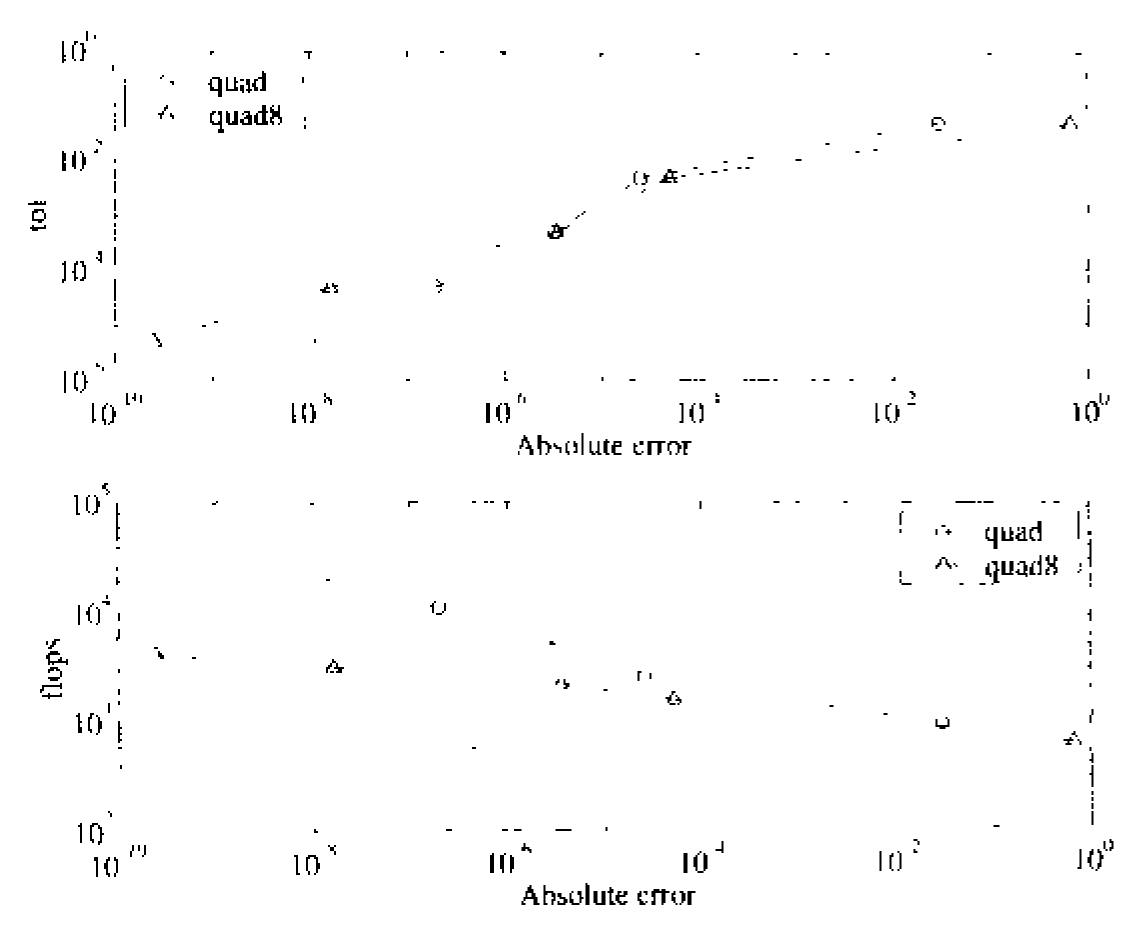


图11-17　内置quad和quad8函数对一维humps函数积分的性能图　图示由demoQuad函数产生

程序清单11-11　函数demoQuad通过对方程(11-43)中的humps函数的积分示范了函数quad和quad8的使用

```
function demoQuad(a,b)
% demoQuad  Use built in quad and quad8 to integrate 'humps' on [0,2]
%
% Synopsis:  demoQuad
%            demoQuad(a,b)
%
% Input:  a,b = (optional) upper and lower limits of integral
%               Default:  a = 0;   b = 2
%
% Output: Tables of absolute error and flops as function of tol parameter
%         input to quad and quad8.  Plots of error vs. tol and error vs. flops

if nargin<2,  a=0; b=2;  end

tol = [5e-2  5e-3  5e-4  5e-5];     %  sequence of relative tolerances
for k = 1:length(tol)
  flops(0);  q(k)  = quad('humps',0,2,tol(k));   f(k) = flops;
  flops(0);  q8(k) = quad8('humps',0,2,tol(k));  f8(k) = flops;
end
Iexact = humpInt(0,2);  e = abs(q - Iexact);  e8 = abs(q8 - Iexact);

fprintf('                 ----- quad -----      ----- quad8 -----\n');
fprintf('    tol          error       flops       error        flops\n');
for k = 1:length(tol)
  fprintf('%11.2e  %11.2e  %6d   %11.2e   %6d\n',tol(k),e(k),f(k),e8(k),f8(k));
end

subplot(2,1,1);
loglog(e,tol,'o--',e8,tol,'^-');
```

657
658

```
legend('quad','quad8',2);  xlabel('Absolute error');  ylabel('tol');

subplot(2,1,2);
loglog(e,f,'o--',e8,f8,'^-'),
legend('quad','quad8');   xlabel('Absolute error');   ylabel('flops');
```

虽然quad和quad8函数对积分估计都有了改进，但是由demoQuad的表格输出和图11-17的下图可知quad8的估计更有效率。在相同的容差水平下，quad8所用的浮点操作次数明显地比quad的要少。原因是quad8收敛到容差范围内所调用$f(x)$的次数比quad的少。

11.4.3 新的quad和quadl函数

在本书出版时，MATLAB 6正在酝酿中，它将会有两个新的自适应求积程序，它们基于Gander和Gautschi[25]新近开发的函数adaptsim和adaptlob。这两个新的内置函数就是quad和quadl，将代替原来的quad和quad8函数。新quad仍然基于Simpson公式，但它包含了改进的停止准则和Romberg外插法步骤。停止准则就是要产生积分的一个快速初始近似，以确定细分容差的数量级。这种办法加上Gander和Gautschi介绍的其他停止准则细节，可有助于防止自适应算法陷入局部容差不能满足的子区间中，但是这不会对积分量值有很大的影响。

函数quadl使用与新函数quad相同的改进停止准则。此函数中的积分算法基于四节点的Gauss-Labatto公式，是对三节点Kronrod的扩展。Kronrod的第二次扩展提供了控制自适应细分的误差估计。Kronrod技术是对基本高斯求积法的修改，以使其对自适应细分更有效。可以回想一下，高斯求积法使用的是一个小段上的不等距节点，而Kronrod扩展选择节点的位置以使高斯公式得到最优化的精度，并且为下一细分级提供了尽可能多的公共节点。Gander和Gautschi已经指出，新的quad和quadl函数比原来的quad和quad8函数更有效。新函数也明显地更可靠，它测试了大量的积分，能够求出在用户自定义容差内的积分值。注意，自适应求积方法不可能保证绝对满足用户自定义的容差，因为收敛准则是基于对截断误差的估计——真正的积分值一般情况下是未知的。

新的quad和quadl函数和原来的（版本5及以前的）quad和quad8函数有相同的输入和输出参数。当新的MATLAB 6发行后，应使用新的quad和quadl函数来代替原来的quad和quad8函数。 659

11.5 广义积分和其他复杂问题

特殊积分或者有一个或多个无穷极限，或者在积分范围内或积分极限上有奇异点。它的一般的特性是被积函数值或积分极限无界。这样的积分一般要先通过一种或多种数学变换来去掉无界点，然后再使用数值方法来求解。各种复杂问题有各自特定的变换，这样就没有一般公式，只有一些计算方法。

数值求积的另一个复杂问题是被积函数的函数值在正负之间振荡，这种问题不是无界的，但仍然会由于正负值抵消而丢失精度。

Davis和Rabinowitz[13]讨论了处理这些问题的不同方法。QUADPACK程序提供了处理特殊积分的几种策略，包括使用坐标变换、特殊权函数和不同基本求积公式的组合。本节中，我们只介绍现有技术中的一小部分。

无穷积分

本节考虑形式为

$$\int_a^{\infty} f(x)\,\mathrm{d}x$$

的积分。下限a是一个有限标量，且一般$a = 0$。只有当$f(x)$有界且x趋向无穷时它趋于零，积分才可求解。求解这种积分明显的困难是上限不能用有限精度的算术式表示。一种快速的解决方法是用一个很大的数x^*（x^* < realmax）来代替无穷大作为上限，但是这种方法的一个问题是一般情况下积分计算起来非常困难。

无穷极限逼近 有无穷上限的积分可写为：

$$I = \int_0^{\infty} f(x)\,\mathrm{d}x = \int_0^{z_1} f(x)\,\mathrm{d}x + \int_{z_1}^{z_2} f(x)\,\mathrm{d}x + \int_{z_2}^{z_3} f(x)\,\mathrm{d}x + \ldots \tag{11-44}$$

其中$z_1<z_2<z_3<\ldots$，为方便起见，令

660

$$I_j = \int_{z_{j-1}}^{z_j} f(x)\,\mathrm{d}x \tag{11-45}$$

故方程（11-44）变成

$$I = \lim_{k \to \infty} \sum_{j=1}^{k} I_j$$

当x的部分积分项的积分值（I_j）对I的影响较小时，就可能简化数值逼近的过程。最简单的简化就是截取部分积分的和。如果间隔大小增加，就可以额外节省计算工作。例如，可以选择

$$z_{j+1} - z_j = \rho(z_j - z_{j-1}) \tag{11-46}$$

其中$\rho > 1$是控制间隔增长率的因子。此方法经整理后，就是当满足

$$|I_{k+1}| < \delta \left| \sum_{j=1}^{k} I_j \right|$$

时停止对间隔相加，这里δ是相对容差。这些思想都融汇在程序清单11-12中的quadToInfinity函数中。部分积分I_j可用任何有限积分的数值求积方法来计算，函数quadToInfinity允许用户在gaussQuad，quad和quad8三种方法之间选择，其他方法可通过修改quadToInfinity的源代码轻易地添加进去。

例11.17 quadToInfinity**的应用**

下面的MATLAB程序段使用quadToInfinity函数来计算如下积分。

$$I = \int_0^{\infty} \exp(-x^2)\,\mathrm{d}x = \frac{\sqrt{\pi}}{2}$$

```
>> fun = inline('exp(-x.^2)');
>> I = quadToInfinity(fun);

j        dx        x2       I_j           Isum
0        0.5       0.5      0.46128101    0.46128101
1        1.0       1.5      0.39490739    0.85618839
2        2.0       3.5      0.03003787    0.88622627
3        4.0       7.5      0.00000066    0.88622693
4        8.0      15.5      0.00000000    0.88622693
5       16.0      31.5      0.00000000    0.88622693
6       32.0      63.5      0.00000000    0.88622693
```

```
>> err = I-sqrt(pi)/2
err =
   1.1102e-16
```

661

程序清单11-12 函数quadToInfinity用来计算无穷上限的积分

```
function Isum = quadToInfinity(fun,a,dx0,tol,method)
% quadToInfinity  Integral from a to infinity evaluated as sum of integrals
%                 Size of subintervals increases geometrically.  Sum is
%                 terminated when change in integral is less than tolerance
%
% Synopsis.  I = quadToInfinity(fun)
%            I = quadToInfinity(fun,a)
%            I = quadToInfinity(fun,a,dx0)
%            I = quadToInfinity(fun,a,dx0,tol)
%            I = quadToInfinity(fun,a,dx0,tol,method)
%
% Input:  fun = (string) name of m-file that evaluates the integrand, f(x)
%         a   = (optional) lower limit of the integral.  Default:  a = 0
%         dx0 = (optional) size of first interval of x axis used to
%               evaluate the partial integrals.  Default:  x = 0.5
%         tol = (optional) relative tolerance.  Sum is terminated when
%               abs(I_{k+1}/I_k) < tol, where I_k - sum(I_j), j=1,...,k.
%               and I_j is integral over subinterval j.  Default: tol=5e-4
%         method = (optional) integral rule used for subintervals. method=1
%               for composite Gauss-Legendre quadrature with 8 panels and 6
%               nodes per panel; method=2 for built-in quad; method=4 for
%               built-in quad8.  Default:  method = 1
%
% Output:  I = estimate of the integral of f(x) dx from x=a to x=infinty
if nargin<2,      a = 0;  end
if nargin<3,  dx0 = 0.5;  end
if nargin<4, tol = 5e-4;  end
if nargin<5, method = 1;  end
j = 0;  dx = dx0;   Isum = 0;   x1 = a;   maxint = 35;   %  Initialize

fprintf('\n    j       dx       x2       I_j           Isum\n');
while j<maxint
  x2 = x1 + dx;
  switch method
    case 1,     I = gaussQuad(fun,x1,x2,8,6);
    case 2,     I = quad(fun,x1,x2);
    case 3,     I = quad8(fun,x1,x2);
    otherwise,  error(sprintf('method = %d not allowed',method));
  end
  Isum = Isum + I,
  fprintf(' %4d %8.1f %8.1f  %12 8f   %12.8f\n',j,dx,x2,I,Isum);
  if j>5 & abs(I/Isum) < tol,  break;  end
  j = j + 1;  x1 = x2;   dx = 2*dx;  %  prepare for next interval
end
```

662

被积函数随着x快速衰减，所以几乎不需要子区间。另外，采用dx0的缺省值时，高斯求积方法在$x = 0$附近的精度很高。

一个更复杂的积分是（见文献[59,第84页，问题16]）

$$I = \int_0^{\infty} \frac{x^{1/5}}{(1+10x)^2}\,\mathrm{d}x = \frac{-10^{-6/5}\pi}{5\sin(6\pi/5)} = 0.0674467742$$

下面的MATLAB程序段使用quadToInfinity函数在缺省参数下计算此积分值。

```
>> fun = inline('x.^(1/5)./(1+10*x).^2');
>> I = quadToInfinity(fun)

  j       dx        x2        I_j          Isum
  0      0.5       0.5     0.04896325   0.04896325
  1      1.0       1.5     0.00998439   0.05894765
  2      2.0       3.5     0.00405910   0.06300675
  3      4.0       7.5     0.00201032   0.06501707
  4      8.0      15.5     0.00107734   0.06609441
  5     16.0      31.5     0.00059835   0.06669275
  6     32.0      63.5     0.00033803   0.06703078
  7     64.0     127.5     0.00019256   0.06722334
  8    128.0     255.5     0.00011015   0.06733348
  9    256.0     511.5     0.00006313   0.06739661
 10    512.0    1023.5     0.00003622   0.06743284
 11   1024.0    2047.5     0.00002079   0.06745363
I =
    0.0675

>> Iexact = 10^(-6/5) * (-1/5)*pi/sin(pi*6/5)
Iexact =
    0.0674

>> err = I-Iexact
err =
   6.8584e-06
```

练习29的目标就是改进此积分的数值逼近。

Gauss-Laguerre求积法　Laguerre多项式是定义在$[0,\infty)$上、权函数为$w(x)=e^{-x}$的正交多项式（参考方程(11-22)，回想一下Gauss-Legendre求积法中的$w(x)=1$）。当用Laguerre多项式来定义高斯求积方法中的节点和权时，就得到Gauss-Laguerre公式

$$\int_0^{\infty} e^{-x} f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i f(x_i) \tag{11-47}$$

[663]

其中x_i是n阶Laguerre多项式的零点，c_i是相应的权值。Gauss-Laguerre的节点-权值表已经建立起来（例见[1]）。一旦c_i和w_i已知，Gauss-Laguerre方法的实现就很简单了。注意，Gauss-Laguerre没有复合公式，因为节点和权都定义在整个$[0,\infty)$区间上。

在一些被积函数中，没有明确的权函数e^{-x}，这样的积分可以写成

$$\int_0^{\infty} f(x)\,\mathrm{d}x = \int_0^{\infty} e^{-x} e^{x} f(x)\,\mathrm{d}x \approx \sum_{i=1}^{n} c_i e^{x_i} f(x_i) \tag{11-48}$$

换言之，用修改的权$c_i e^{x_i}$代替了方程（11-47）中的c_i。Davis和Rabinowitz[13]警告说，方程（11-48）只能在可用某个多项式很好地逼近$e^x f(x)$时才可以使用。

作为一个计算公式，方程（11-48）非常有用，它也表明Gauss-Laguerre求积法使用快速衰减的$f(x)$可能会更成功（有关Gauss-Laguerre求积法的其他信息，可参见[13]）。

方程（11-47）和方程（11-48）中的公式在程序清单11-13中的gaussLagQuad函数加以

实现 节点和权值作为GlagTable中的查找表提供，或者在GlagNodeWt中通过求解一个相关的特征值问题来得到，GlagTable和GlagNodeWt在此处未列出，它们包含在NMM工具箱的integrate目录中。

例11.18 Gauss-Laguerre求积法的应用

在本例中，使用gaussLagQuad函数，由两个测试积分来示范Gauss-Laguerre求积法。第一个测试积分是

$$I = \int_0^\infty x^4 e^{-x}\,dx = 24$$

下面的语句依照方程(11-47)定义了$f(x)$，并使用升序的Gauss-Laguerre求积法计算：

```
>> fun = inline('x.^4');
>> I = gaussLagQuad(fun,2)
I =
   20.0000

>> I = gaussLagQuad(fun,3)
I =
   24.0000
>> I = gaussLagQuad(fun,4)
I =
   24.0000
```

664

程序清单11-13 函数gaussLagQuad使用Gauss-Laguerre求积法来计算形式为 $\int_0^\infty e^{-x} f(x)dx$ (wtype=1)或 $\int_0^\infty f(x)dx$ (wtype=2)的积分

```
function I = gaussLagQuad(fun,nnode,wtype,varargin)
% gaussLagQuad  Gauss-Laguerre quadrature for integrals on [0,infinity)
%
% Synopsis:  I = gaussLagQuad(fun,node)
%            I = gaussLagQuad(fun,node,wtype)
%            I = gaussLagQuad(fun,node,wtype,arg1,arg2,...)
%
% Input:     fun   = (string) name of m-file that evaluates f(x)
%            nnode = number of nodes on each subinterval
%            wtype = (optional) flag indicating how weight function is applied
%                    If wtype = 1 (default) the integrand is of the form
%                    exp(-x)*f(x) and the quadrature rule is sum(w(i)*f(x(i))).
%                    If wtype = 2 the integrand is of the form f(x)
%                    and the quadrature rule is sum(w(i)*exp(x(i))*f(x(i))).
%            arg1, arg2 = (optional) parameters passed through to fun
%
% Output:    I = approximate value of the integral

if nargin<3,  wtype = 1,  end

if nnode<=25
  [x,w] = GLagTable(nnode);         %  Look up nodes and weights,
else                                %  or if n is too big for the table,
  [x,w] = GLagNodeWt(nnode);        %  compute the nodes and weights
end

f = feval(fun,x,varargin{:});
```

```
if wtype == 1
  I = w'*f;              %  int(exp(-x)*dx) = sum(w*f)
else
  we = w.*exp(x);        %  Use rule in the form int(f*dx) = sum(w*exp(x)*f)
  I = we'*f;             %  not int(exp(-x)*dx) = sum(w*f)
end
```

积分的精确值由三阶公式得到，这与高斯求积法的理论是一致的。对于由合适的权函数（Laguerre多项式的为e^{-x}）定义的正交多项式来说，n阶的高斯求积公式可得到$2n-1$阶多项式的精确积分值。这个测试函数使我们有信心认为，gaussLagQuad函数正确地实现了Gauss-Laguerre求积法。

第二个测试积分是

$$I = \int_0^\infty \exp(-x^2)\,\mathrm{d}x = \frac{\sqrt{\pi}}{2}$$

它在例11.17中求解过。既然e^{-x}不是被积函数中的一个因子，那么就必须使用方程(11-48)的公式。这受$f(x)=e^{-x^2}$和wtype = 2的影响，将这二者作为gaussLagQuad的输入，得：

```
>> fun = inline('exp(-x.^2)');  Iexact = sqrt(pi)/2;
>> I = gaussLagQuad(fun,5,2),    err = I-Iexact
I =
    0.8558
err =
   -0.0304

>> I = gaussLagQuad(fun,15,2),   err = I-Iexact
I =
    0.8864
err =
   1.2449e-04

>> I = gaussLagQuad(fun,25,2),   err = I-Iexact
I =
    0.8862
err =
   3.7225e-06
```

误差随着Gauss-Laguerre求积公式阶数的增加而减小，但是速度并不快，原因是$f(x)=e^{-x^2}$不能用多项式很好地逼近。使用quadToInfinity函数可以更容易地得到这个积分的精确值。

11.6　小结

本章介绍了几种基本和复合数值求积公式。基本公式是构建复合公式的基础，而复合公式在任意的有限区间上逼近积分值。表11-1列出了本章开发的m文件函数的名字。

表11-1　本章开发的数值积分函数。小节中带N.A.（没有使用）的没有在文章中列出，但包含在NMM工具箱中

函　数	小　节	描　述
adaptSimpson	651	基于Simpson公式的自适应数值积分
compIntRules	N.A	比较梯形公式、Simpson公式和Gauss-Legendre求积公式的计算效率

（续）

函数	小节	描述
demoGauss	641	使用Gauss-Legendre求积公式来计算erf (x)
demoGaussLag	N.A.	使用Gauss-Legendre求积公式来计算$[0,\infty)$上的两个积分
demoSimp	N.A.	通过对$f(x)=xe^{-x}$的积分示范Simpson公式的应用
demoTrap	608	通过对$f(x)=xe^{-x}$积分来示范梯形公式的应用
expmx2	641	计算$\exp(-x^2)$，其中x是一个标量或是一个向量，常用来计算erf (x)
gaussLagQuad	665	在$[0,\infty)$上使用Gauss-Legendre求积法对用户自定义函数积分
gaussQuad	640	对用户自定义函数使用复合Gauss-Legendre求积法积分
GLagTable	N.A.	查询阶数低于15的Gauss-Legendre求积法的节点和权值
GLTable	630	查询阶数低于8的Gauss-Legendre求积法的节点和权值
GLagNodeWt	N.A.	计算任意阶数的Gauss-Laguerre求积法的节点和权值
GLNodeWt	635	计算任意阶数的Gauss-Legendre求积法的节点和权值
humpInt	N.A.	计算方程(11-43)所定义的内置humps函数的一个定积分的精确值
makeGLtable	N.A.	建立Gauss-Legendre求积法的节点和权值表。输出格式化为能直接复制和粘贴到Gltable.m中的形式
plotSimpInt	N.A.	产生Simpson公式积分的图形描述
plotTrapInt	N.A.	产生梯形公式积分的图形描述
rcursiveIndent	N.A.	示范递归函数调用的应用
simpson	615	使用Simpson公式对用户自定义函数进行积分
trapezoid	607	使用梯形公式对用户自定义函数进行积分
trapzDat	611	使用梯形公式对离散数据进行积分
xemx	608	计算$x\exp(-x)$，其中x是一个标量或者是一个向量

本章举出了几个例子，通过数值检验来展示复合公式的精度特性。对梯形公式、Simpson公式、Gauss-Legendre公式来说，在精度提高的同时，效率也提高，这是因为使用这些算法对被积函数进行的计算较少，即能获得期望的精度。其中，Gauss-Legendre公式是精度最高而且最有效的，梯形公式却恰好相反。

将基本公式与自适应算法结合可得到更高的效率。用户指定一个积分估算误差上的容差，自适应算法以此选择计算被积函数的点，使结果满足容差。内置quad和quad8函数就是实现自适应算法的例子。

在本章的结尾，介绍了一个用特殊技巧计算无穷上限积分的小例子。

667

遇到一个新的数值积分问题时，最明智的方法是尝试多种求积法，并改变控制每个公式的参数。这样计算出积分的一系列估计值，从而更大可能地得到与真实值最接近的估计。对非自适应算法来说，细分策略就是增加用来计算积分的小段数。而对自适应算法而言，细分策略就是在积分数值上对容差有步骤地减小。

补充读物

很多基本的数值分析书籍都涉及数值积分的内容。这方面，介绍性方面的书籍有Cheney和Kincaid [10]以及Burden和Faires[9]。Acton [2]则强调使用数值方法之前对被积函数属性作仔细检查的重要性。对于算法本身的数学分析，可以参考Isaacson和Keller[40]以及Stoer 和Bulirsch[70]。Forsythe等[24]和Kahaner等[43]重点介绍了算法的实现，并特别追溯了quad和quad8的推导过程。Press等[61]提供了这方面的C程序源代码。

Davis和Rabinowitz[13]用Fortran代码对各种数值积分方法进行了综合评测。QUADPACK库[59]实现了单变量定积分计算的各种数值积分方法，它用Fortran代码[⊖]描述。Kahaner等[43]提供了QUADPACK程序的一个子集，介绍了离散数据上的自适应Gauss-Kronrod求积法和三阶分段Hermite积分。感兴趣的读者可以使用NMM工具箱的integrate目录下的adaptGK和demoAdaptGK函数进行实验。函数adaptGK使用的是自适应求积方法，此种算法基于[59]介绍的Gauss-Kronrod 7-15公式。

Krommer和Ueberhuber[48]介绍了数值积分方面最新的发展状况，包括在并行计算机上的实现、符号软件包的使用和二维以上的积分。

习题

668

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1. (1)手工计算下列积分：

$$I_1 = \int_{\pi/2}^{\pi}(x^2 + x + 1)\,\mathrm{d}x, \qquad I_2 = \int_{3}^{5}(x^3 - 1)\,\mathrm{d}x$$

2. * (2)写一个polyInt函数，使用内置的polyval函数来计算多项式的定积分。此函数的输入应该是多项式系数和积分上下限组成的向量。通过计算上题两个积分来检验你的函数。

3. * (1+)使用MATLAB的学生版或者数学符号工具箱中的符号能力来求广义humps函数

$$f(x) = \frac{1}{(x - c_1)^2 + c_2} + \frac{1}{(x - c_3)^2 + c_4} + c_5$$

的定积分。

4. (1+)使用两小段的梯形公式手工（即用铅笔和纸）计算下列积分，并将结果与积分的精确解析值及trapezoid函数得到的结果进行比较。

(a) $I = \int_0^2 x\exp(-x)\,\mathrm{d}x$

(b) $I = \int_0^{\pi}\cos(x)\,\mathrm{d}x$

5. (1+)使用Simpson公式代替梯形公式，重复上题的计算过程。

6. (3)写一个quadSpline函数，来计算三阶样条逼近式的积分；逼近式由程序清单10-11中的spline函数来求出。此函数的序言要求如下：

```
function I = quadSpline(x,y)
% quadSpline   Evaluate integral of a 1D cubic-spline
%
% Synopsis:    I = quadSpline(x,y)
%
% Input:       x,y = vectors of data defining the knots of the spline
%
% Output:      I = integral of cubic-spline passing through knots used
%                  to create the spline.  Uses the splint function.
```

用程序清单4-3中的trapzDatTest函数的修改版来检验你的函数，证明其有效。此函数中α的值是多少？

⊖ 在www.netlib.org中有QUADPACK的源代码。

7. (2) 复制trapezoid.m文件（参见程序清单11-1以及NMM中的相应文件），复制文件称为trapezerr.m文件，并将该复制文件中的最后一行改为

669

```
I = h * (0.5*f(1) + sum(f(2:n)) + 0.5*f(n));
```

也就是说，用f(2:n)来代替f(2:n-1)子表达式，从而在代码中引入了一个bug。修改程序清单11-2中的demoTrap函数，使其由调用trapezoid函数改为调用trapezerr函数。运行修改的demoTrap函数。绝对误差的值会随着n的增加而减小吗？α的值发生了什么变化？如果α没有计算出来，有可能推断出trapezerr函数确实在正确地工作吗？

8. * (2+)F.M.White(*Fluid Mechanics*, fourth edition, 1999，McGraw-Hill，NewYork，problem 6.57)给出了下列圆管中速度分布数据：

r/R	0.0	0.102	0.206	0.412	0.617	0.784	0.846	0.907	0.963
u/u_c	1.0	0.997	0.988	0.959	0.908	0.847	0.818	0.771	0.690

r是径向位置，R = 12.35 cm为管道的半径，u是在位置r处的速度，u_c是在轴心线r = 0处的速度。圆管中的平均速度定义为

$$V=\frac{1}{\pi R^2}\int_0^R u2\pi r\,\mathrm{d}r,\quad 或\quad \frac{V}{u_c}=\int_0^1 2\frac{u}{u_c}\eta\,\mathrm{d}\eta$$

其中$\eta = r/R$。如果已知u_c = 30.5m/s，则V的值是多少？别忘了在r/R = 1时，有u/u_c = 0这个隐含条件。表格中的数据包含在NMM工具箱的data目录下的vprofile.dat文件中。

9. (2)使用trapezoid函数在依次增长的n（减小的h）上计算erf (1)。为使绝对误差小于5×10^{-8}，h的值应该为多少？

10. (2)使用simpson函数代替trapezoid函数重复练习9中的计算。

11. (2)使用gaussQuad函数代替trapezoid函数重复练习9中的计算。

12. * (2+)对任意的m和n以及依次减小的小段大小h，使用梯形公式计算

$$\beta(m,n)=\int_0^1 x^{m-1}(1-x)^{n-1}\,\mathrm{d}x$$

打印出数据$\beta(m, n)$及其与内置的beta函数返回值的相对误差。使用自己设计的函数计算$\beta(1,2)$、$\beta(1.5,2.5)$、$\beta(2,3)$和$\beta(2,5)$，并评价收敛速率。（提示：m和n的值可以用全局变量在trapezoid函数中传送（不能传出去））。

13. (2)使用Simpson公式重复练习12中的计算。

14. (2)使用Gauss-Legendre求积法重复练习12中的计算。

15. (2+)写一个m文件返回

$$p(s)=\frac{1}{\sqrt{2\pi}}\int_{-s}^{s}\mathrm{e}^{-z^2/2}\,\mathrm{d}z$$

的值，使用6个小段、每个小段4个节点的复合Gauss-Legendre求积法。$p(s)$是在服从方差为1和均值为0的标准正态分布的随机群体中样本落在$\pm s$之间的概率。对s = 1,s = 2,s = 2.5和s = 3，分别计算$p(s)$。

670

16. * (2)使用NMM中的程序trapezoid，simpson和gaussQuad来计算下式

$$I=\int_0^1\sqrt{x}\,\mathrm{d}x$$

对每个程序，计算至少三种不同小段数的积分。作一个表格，分别描述所测量截断误差在不同小段数下的变化。报告在求I值过程中的任何问题，在这个问题上哪个程序工作得最好？

17. (2+)若$w(x)=1$或$w(x)=(1-x^2)^{-1/2}$，证明（见11.3.1节）$g(x)=1$和$h(x)=x$在$(a,b)=(-1,1)$正交。

18. (3)正交多项式组$\tilde{\Phi}_i(x)$定义为

$$\left\langle\tilde{\Phi}_i(x),\tilde{\Phi}_j(x)\right\rangle=\int_a^b w(x)\tilde{\Phi}_i(x)\tilde{\Phi}_j(x)\,\mathrm{d}x=\delta_{ij}=\begin{cases}0, & i\neq j\\ 1, & i=j\end{cases}$$

已知正交多项式组$P_i(x)$，我们可以将每个$P_i(x)$乘以一个合适的常数，以使前面的条件成立。注意在决定常数时只有$i=j$的情况是重要的。

(a) 已知11.3.1节末的Legendre多项式，找出前三个正交Legendre多项式。

(b) 使用高斯求积法证明你在(a)中的推断满足条件$i=j$。

19. (2+)由例11.11中的计算，求出3阶Gauss-Legendre公式的节点和权值。

20. (2+)写一个m文件函数，使用小段数增加、每个小段的节点数固定的复合Gauss-Legendre求积公式来计算$\int_0^5 x\exp(-x)$。当小段数分别为1、2、3、4、6和8个时，两节点Gauss-Legendre公式的截断误差表达式$O(h^\alpha)$中的α值是多少？比较计算值与截断误差理论值。

21. (3)使用四节点和八节点的Gauss-Legendre公式重复前个练习。讨论你在计算α中遇到的任何问题。

22. * (2+)写一个m文件函数，使用每小段两节点的复合梯形公式、复合Simpson公式和复合Gauss-Legendre公式计算$\int_0^{2\pi}\sin^2(x)\,\mathrm{d}x$。将`trapezoid`，`simpson`和`gaussQuad`的调用放在一个循环中，对`np = [1 2 4 8 16 32]`重复以上计算，其中np是小段数。记录每种方法的函数计算次数n，打印出三种方法的绝对误差$|I-I_{\text{exact}}|$和n（参见[13.2.9节]中的例子对结果的解释会很有帮助）。

23. (2)(Kahaner等[43])用下列积分来估计π的值

$$\pi=\int_0^1\frac{4}{1+x^2}\,\mathrm{d}x$$

671

比较2、4、8、16、32、64和128个小段的复合梯形公式、复合Simpson公式和复合Gauss-Legendre公式的收敛特性。

24. (2)选择一种积分方法，写出必要的MATLAB代码对任何k值计算椭圆积分

$$E=\int_0^{2\pi}\sqrt{1-k^2\sin^2 x}\,\mathrm{d}x$$

比较你的结果和由`ellipke`函数在$a=2$，$b=1$以及$a=12$，$b=1$时产生的结果。函数`ellipke`在求积分值时使用了什么技巧？

25. (2+)通过计算

$$I=\int_0^1\sqrt{x}\,\mathrm{d}x$$

来检验自适应求积程序`adaptsimpson`，`quad`和`quad8`（或`quadl`）的性能。对每个程序，至少要在三个不同的容差下计算积分。列出一个表，比较测出的截断误差作为容差的函数的情况。报告在求I值中出现的任何问题。在这个问题上哪个程序工作得

最好？将这些自适应程序的特性与练习16中的非自适应算法比较。

26. (2+)（假设你使用MATLAB 6）修改程序清单11-11中的demoQuad函数，比较新函数quad和quadl在计算内置的humps函数积分时的性能。新的quad和quadl性能之间的关系与原来的quad和quad8性能之间的关系类似吗？

27. (3) 一个物体的热辐射量（thermal radiation）是其热力学温度的函数。黑体发射器（blackbody emitter）是一个理想的表面，它在所有方向同等地进行热辐射，并且能吸收入射于其表面的所有辐射（例见F.P.Incropera和D.P.Dewitt，*Fundamentals of Heat and Mass Transfer*, fourth edition, 1996, Wiley, NewYork.）。Planck分布

$$E_{\lambda,b} = \frac{c_1}{\lambda^5[\exp(c_2/\lambda T)-1]}$$

描述了黑体发射器发射功率变化关于波长的函数。其中λ是波长，单位为μm，T是热力学温度，单位为K，$c_1 = 3.7418 \times 10^8 \mu m^4/m^2$，$c_2 = 1.4388 \times 10^4 \mu m \cdot K$，波长在$0 \leqslant \lambda \leqslant \lambda^*$的发射能量为

$$F_{0-\lambda^*} = \int_0^{\lambda^* T} \frac{E_{\lambda,b}}{\sigma T^5} d(\lambda T)$$

其中$\sigma = 5.6696 \times 10^{-8} W/(m^2K)$。$F_{0-\lambda^*}$定义的被积函数依赖于乘积$\lambda T$，而非分别依赖于$\lambda$和$T$。

写出MATLAB函数对任何λT的输入值计算$F_{0-\lambda^*}$。对$\lambda^* T$ = 1000、5000、8000、10000和20000时计算$F_{0-\lambda^*}$（答案：在$\lambda^* T$ = 5000时有$F_{0-\lambda^*}$=0.633766）。注意，$F_{0-\lambda^*}$的计算值决定于常数c_1、c_2和σ的数值。$F_{0-\lambda^*}$函数与其他表格数据不同，它更依赖于定义$F_{0-\lambda^*}$的常数，而不是积分算法。虽然如此，还是要尽可能提高积分数值计算中的精度。

28. (2+)使用数值方法计算下列积分值：

(a) $I = \int_0^{\infty} \frac{2\,dx}{1+x^2} = \pi$

(b) $I = \int_0^{\infty} x^2 e^{-x}\,dx = 2$

(c) $I = \int_0^{\infty} \frac{\ln(x)\,dx}{1+100x^2}\,dx = \frac{-\pi}{20}\ln(10)$

29. (2+)改变quadToinfinity的输入参数，找出对例11.17第二个积分的更精确逼近。哪一个选项使精度最高？

30. (3+)Stanley Middleman（*An Introduction to Mass and Hrat Transfer*, 1998, Wiley, NewYork, p.221）分析了管道中流动溶剂对固体残留物的溶解问题。为找出溶解给定厚度的残留物需要的时间，需要找到满足下式的τ值。

$$\int_0^{\tau} \frac{du}{(1-u^3)^{1/3}} = 1$$

将上面的方程写为：

$$f(\tau) = \int_0^{\tau} \frac{du}{(1-u^3)^{1/3}} - 1$$

将此问题转化为一个求根问题，即求出使$f(\tau) = 0$的τ值。使用内置的fzero函数和

672

一种自适应求积方法解此问题。注意求$f(\tau)$的m文件需要输入τ值向量并返回$f(\tau)$值向量。实现这种功能的一种简单的方法就是使用与如下代码类似的代码:

```
f = zeros(size(tau));
for k=1:length(tau)
    f(k) = ...            %  evaluate f( tau(k) )
end
```

[673]

满足积分方程的τ为多少? 提示: $\tau<1$。

第12章 常微分方程的数值积分

微分方程是描述一个变量关于另一个变量的变化率的数学模型。很多基本的物理定律、包括质量、动量和能量的守恒定律、都自然地表示为微分方程。微分方程有很多种，相应地也有特定的求解过程。本章讨论初值问题，已知微分方程和合适的初始条件，解初值问题可得到因变量合适的表达式，初值问题在电路分析、机械动力学、热传输、化学动力学、人口动力学、经济学以及科学技术的其他领域中都有应用。

本章主题

1. **基本思想和术语**

   本节定义常微分方程（Ordinary Differential Equation，ODE）和初值问题的原型，另外还描述了ODE精确（解析）解和数值解的显著特征。

2. **欧拉法**

   本节推导、分析并用MATLAB实现简单的欧拉法，用数值实验推导和证明了局部离散误差和全局离散误差的表达式。欧拉法是最简单的单步方法，在时间t_k处的因变量只根据前一步的因变量来计算。

3. **高阶单步法**

   这里介绍了中点法、Heun法和四阶Runge-Kutta法。离散误差表达式的数量级以及一些数值实验都说明了它们比欧拉法更有效。

4. **自适应步长法**

   自适应步长算法调整自变量的变化，以使对ODE数值解的估计落在用户自定义容差内。内置的ode23和ode45函数就使用了自适应步长算法。然后介绍了此算法包含的基本思想，并用例子示范了ode23和ode45函数的用法。

5. **联立ODE组**

   很多实际的初值问题涉及联立ODE组和高阶ODE，高阶ODE等价于一阶联立ODE组。这里示范了对一阶联立ODE组的数值解，以及高阶ODE向一阶ODE组的转化方法。

6. **附加主题**

   这里简要介绍前面没有介绍的数值技术，确定了刚性方程的内置算法。

图12-1 第12章的主题

674 ≀ 675

本章讨论的主题总结如图12-1。我们会首先介绍微分方程的一些例子，作为学习数值解算法的前导，然后比较详细地介绍常微分方程最简单的数值解法——欧拉法，给出了欧拉法的MATLAB实现，并分析了其离散误差。接下来介绍更精确的中点法和Heun法，再接下来是四阶Runge-Kutta法，这些方法都作为欧拉法的扩展进行推导。欧拉法、中点法、Heun法和Runge-Kutta法是MATLAB内置ODE积分程序ode23和ode45的基础，这两个程序通过控制自适应步长得到更精确的解。经过了解本章开始介绍的简单程序，读者可掌握精确度以及输入输出参数方面的问题，这些问题将应用到内置自适应步长程序中。

在本章的第二部分，单个常微分方程的积分程序扩展为常微分方程组的积分程序。在四阶Runge-Kutta算法中首先介绍方程组积分的一般过程。这里介绍两个一阶ODE联立的例子——两物种的猎食者-猎物模型，此方程组的求解过程由内置ode45函数来实现。接下来给出单个高阶ODE转化成一阶ODE组来进行积分的程序，这里使用了模拟二阶弹簧减震器系统的例子。然后描述求解任意高阶ODE的通用程序。最后，本章在研究ODE的数值解的过程中简单讨论了附加的一些其他问题。

12.1 基本思想和术语

本节简单介绍了常微分方程，并对本章介绍的数值算法作了概述，此外还定义了一些常用的术语

12.1.1 常微分方程

一个常微分方程的原型系统是：

$$\frac{dy}{dt}=f(t,y) \tag{12-1}$$

其中$f(t,y)$是自变量t和因变量y的函数，dy/dt是常导数（ordinary derivative），因为y只是t的函数。可以将此方程与描述两个自变量以上的偏微分方程进行比较，例如$u=u(x,y)$。

方程（12-1）的比率关系并不能完全确定y。实际上，方程（12-1）有无穷多个解。为确定惟一的$y(t)$，需要提供初始条件：

[676]

$$y(t_0)=y_0$$

前面两个方程经常写在一起：

$$\frac{dy}{dt}=f(t,y),\quad y(t_0)=y_0 \tag{12-2}$$

方程（12-2）就涉及到初值问题(initial-value problem)。为使这个问题更完整，我们也必须确定t的范围（例如，$t_0<t<t_N$），在该范围内对方程进行求解。

在本章中，自变量标识为t，经常将其解释为时间。的确，很多初值问题都是瞬间行为，所以t就自然地被选为自变量。读者要注意，也有很多初值问题不是依赖于时间的，如一个物理问题的自变量可以是x，表示到某参考点的距离，这样初值就是因变量（也可能是它的导数）在物理边界$x=0$处的值。

方程（12-1）称为一阶ODE，因为它只涉及到y的一阶导数。常微分方程可以包含二阶、三阶以及更高阶的导数。一个微分方程的阶就是出现在该方程中因变量导数的最高阶数。

一些ODE也可以与其他ODE联立，这种情况下，每个因变量y_1、y_2、…y_m都有自己的微分方程，并且这些方程也可包含其他因变量表达式。例如，下面定义了两个联立的一阶ODE：

$$\frac{dy_1}{dt}=\alpha y_1+\beta y_2+g_1(t),\qquad y_1(t_0)=y_{1,0}$$

$$\frac{dy_2}{dt}=\gamma y_1+\delta y_2+g_2(t),\qquad y_2(t_0)=y_{2,0}$$

在上面的方程组中，α、β、γ和δ为已知系数，$g_i(t)$为t的已知函数。

本章介绍的所有数值方法都是为解一阶ODE而推导的。如12.5.2节所述，高阶ODE可首先转化为数学意义上等价的联立一阶ODE组，然后通过数值方法求解。

例12.1 牛顿运动定律

对一个物体施加一个外力，其运动表示为

$$F=ma$$

[677] 其中F是力的大小，m是物体的质量，a是物体的加速度。这是一个真正的ODE，因为有

$$\frac{\mathrm{d}v}{\mathrm{d}t}=a$$

其中v是物体的瞬时速度，t为时间。因此，一个表示牛顿运动定律的等价方程就是

$$\frac{\mathrm{d}v}{\mathrm{d}t}=\frac{F}{m} \tag{12-3}$$

方程（12-3）显然具有方程（12-1）的形式。如果$F(t)$和某一时刻t_0的速度已知，就可以将此方程积分求出$v(t)$。在很多有趣的物理问题中，F都是依赖于x、u和其他参数t的。

另外，运动方程也可以用位移x作为因变量（即$x=x(t)$）来表示。由于$u=\mathrm{d}x/\mathrm{d}t$，方程（12-3）等价于

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2}=\frac{F}{m}$$

此二阶ODE需要两个初始条件：

$$x(0)=x_0,\quad \left.\frac{\mathrm{d}x}{\mathrm{d}t}\right|_{t=0}=v_0$$

例12.2 瞬时热传递（牛顿冷却定律）

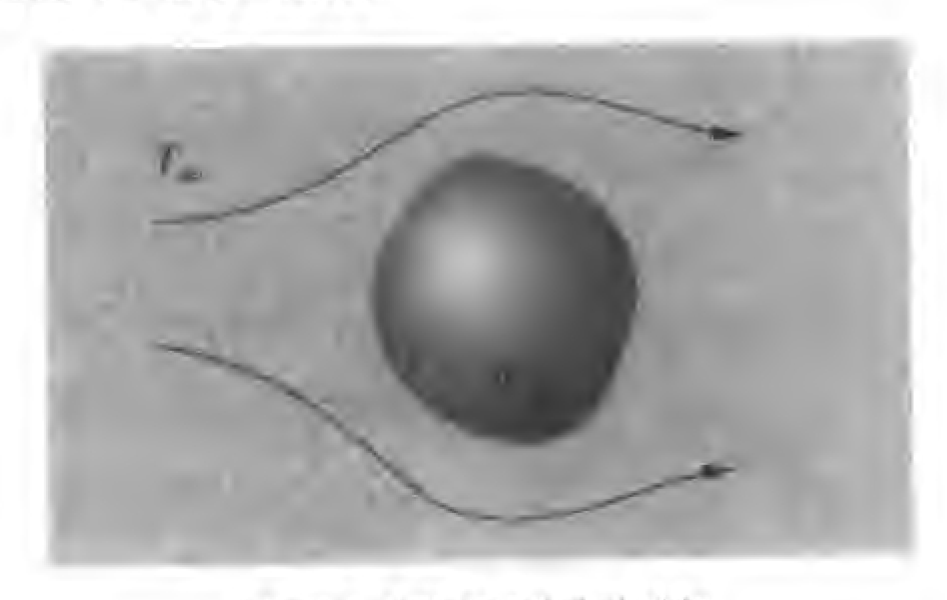


图12-2 物体在流体中的冷却

考虑图12-2中描述的问题。一个质量为m、瞬时温度为T的物体放入流动的液体中，液体的上游温度为T_∞。假设一开始物体的温度与T_∞不同，我们希望知道物体加热或冷却得有多快。牛顿假定物体和液流交换热量的速率与物体表面温度和T_∞的温度差成比例。假设这种比例关系在任何时刻都成立，表示为

678

$$Q=HA\ (T_s-T_\infty)$$

其中Q是热量传输的瞬时速率，H是一个比例常数，称为传热系数。A和T_s分别为物体的表面面积和表面温度。根据物体的能量守恒，有

$$mc\frac{\mathrm{d}T}{\mathrm{d}t}=-Q=-HA(T_s-T_\infty) \tag{12-4}$$

其中c是物体材料的比热。方程（12-4）的左边是物体内部热量增加的速率，右边是热量通过热传递从表面进入物体的速率。很多情况下，物体的导热性足够高，使物体内部对热流的热阻（heat resistance）小于物体表面和流体间的热阻。这样，物体内部的温度将会一致，且表面温度也会与内部温度一致。在这种假设下（即$T_s=T$），方程（12-4）可重写为：

$$\frac{dT}{dt}=-\frac{HA}{mc}(T-T_{\infty}) \tag{12-5}$$

方程现在是方程（12-1）的形式。方程（12-5）也是线性的，可以直接求解（见练习1）。

12.1.2 数值求解策略概述

本章的基本目标可以陈述如下：已知方程（12-2）的初值问题，求出$y(t)$的数值逼近。形式上可写为：

$$y(t)=y_0+\int_{t_0}^{t} f(\tilde{t},y)\,d\tilde{t}$$

于是方程（12-2）的求解可以看成求积分的值。这就是常微分方程数值求解过程被称为ODE数值积分的原因。形式为$dy/dt=f(t)$的ODE可以使用第11章中介绍的方法来进行数值积分。如果$f=f(t,y)$，那么就需要使用本章介绍的方法来求解。

假设方程（12-2）的解存在，但是写不出它的解析式。换言之，对任意给定的y和t，虽然可以求解含dy/dt的公式，但是不能找到一个y作为t的函数的具体公式，该公式能同时满足ODE和初始条件。图12-3描述了方程（12-2）的精确解，以及用某种未确定的方法得到的逼近数值解。初始条件是$y(t)$在t_0的斜率$f(t_0, y_0)$，如图12-3所示。

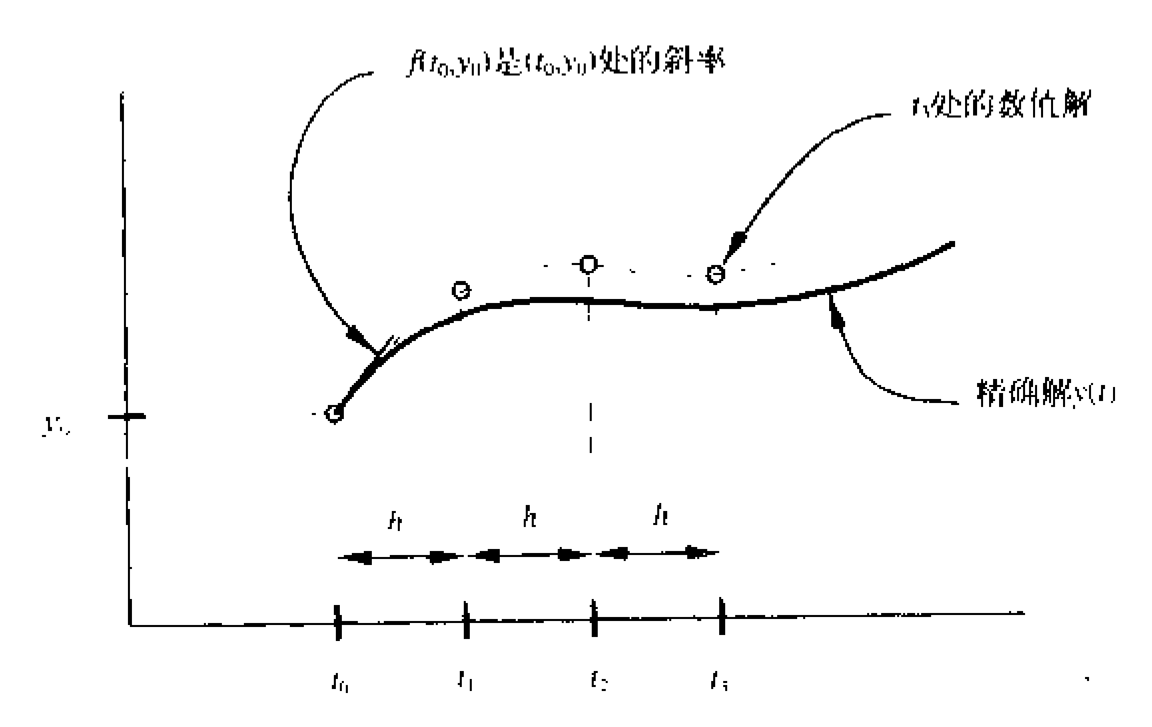


图12-3　对方程$dy/dt=f(t,y)$，$y(t_0)=y_0$精确解数值逼近的初步认识

方程（12-2）只在有限个t值上有数值解。因此，称该数值解是离散的，精确解$y(t)$与之相反，是连续的。将离散t值序列定义为

679

$$t_j=t_0+jh,\qquad j=0,1,2,\cdots,N \tag{12-6}$$

其中距离参数h称为步长。这里，假设h为一个已知常数。在本章后面考虑的更高级的技术中，h将是一个变量。通过在相应的t_j处计算一系列y_j，来得到方程（12-2）的数值解。

本章中，设方程（12-2）的未知精确解标识为$y(t)$，或简称为y，在离散点t_j处的精确解是$y(t_j)$，对$y(t_j)$的数值逼近是y_j，表12-1总结了这些命名法。用这些符号，t_j处数值解的误差可以写为$e_j=y_j-y(t_j)$。

表12-1 常微分方程的数值解法中使用的命名法

符 号	意 义
t_j	自变量t的离散值
$y(t)$	微分方程的精确解
$y(t_j)$	$t = t_j$处的精确解
$e_j = y_j - y(t_j)$	$t = t_j$处逼近解的误差

下标问题 图12-3和方程（12-6）定义离散t值的序列为t_0，t_1，t_2，…，t_N。在计算机程序中，可将t_j存储在一个数组中。但是，因为MATLAB中所有的数组都以索引1而不是0开始，所以不可能将其直接转化成MATLAB代码。粗心的用户在这个细节上就容易犯错。数学符号t_j和MATLAB变量t（j）之间的对应关系总结如下： 680

数学符号：	t_0	t_1	…	t_N
MATLAB变量：	t(1)	t(2)	...	t(n)

以上转换要求：

$$n = N + 1$$

不管用哪种索引符号，h的值都一样。对恒定的h，有

$$h = \frac{t_N - t_0}{N} = \frac{t(n) - t(1)}{n-1} \quad （只对恒定的h）$$

通常，h的值确定，t_j由方程（12-6）计算出来。在MATLAB中，均匀间隔向量t中的元素可由下列语句计算得到：

```
tn = ...          % Specify the final value of t
h = ...           % and the stepsize
t = (0:h:tn)';
```

其中将（0:h:tn）转置产生一个列向量。

12.2 欧拉法

在点$t = t_0$处方程（12-2）未知解的泰勒级数展开式为：

$$y(t) = y(t_0) + (t - t_0)y'(t_0) + \frac{(t-t_0)^2}{2}y''(t_0) + \cdots \tag{12-7}$$

其中$y' = dy/dt$、$y'' = d^2y/dx^2$，依此类推。只保留一阶导数项，用方程（12-1）右端计算$y'(t_0)$，方程（12-7）变为

$$y(t) \approx y(t_0) + (t - t_0)f(t_0, y_0) \tag{12-8}$$

于是，使用上述公式对$y(t_1)$的数值逼近为

$$y_1 = y_0 + hf(t_0, y_0) \tag{12-9}$$

681

其中$h = t_1 - t_0$。图12-3用图形对此方程作了描述。已知$y_0 = y(t_0)$，$y(t_1)$的逼近解可由点(t_0, y_0)处的曲线$y(t)$斜率进行外插得到。由图12-3易知此逼近解的精度受h的大小和$y(t)$函数在$t = t_0$和$t = t_1$间曲率的影响。这些问题在12.2.2节中讨论。

方程（12-9）给出了由已知的t_0、y_0和h来计算y_1的确切公式。既然可以得到y_1的逼近数值解，就可以由同样的过程计算y_2，即

$$y_2 = y_1 + hf(t_1,\ y_1)$$

依次类推。一般地，有

$$y_j = y_{j-1} + hf(t_{j-1}, y_{j-1}), \quad j = 1,2,\cdots,N \tag{12-10}$$

这种简单的积分策略被称为欧拉法，或简易(显式)欧拉法(Euler's explicit method)。说它简易是因为下一步的y值仅仅由前一步的y值计算就可出来。已知逼近公式，就可以由已知的t_{j-1}、y_{j-1}和$f(t_{j-1}, y_{j-1})$很容易地求出y_j。

方程（12-10）是一个逼近法，因为它是由方程（12-7）中的泰勒级数截断而得到的。因此，使用方程（12-10）得到的数值解与精确解之间还有差别。在给出一个欧拉法的计算例子和MATLAB一般的实现之后，还会对此逼近法进行详细的分析。

例12.3　使用欧拉法进行手工计算

本例中将使用初值问题

$$\frac{dy}{dt} = t - 2y, \quad y(0) = 1 \tag{12-11}$$

来测试欧拉法。其精确结果为

$$y = \frac{1}{4}[2t - 1 + 5e^{-2t}]$$

由于精确解已知，似乎就没有必要求数值解，但是，求解此问题便于我们研究数值方法的特性，同时它还提供了一个检查计算机程序bug的基准。

682

下表给出了在$h=0.2$时此问题求解的前几步手工计算结果。作为比较，也列出了精确值$y(t_j)$和误差$y_j - y(t_j)$。

			欧拉法	精确解	误差
j	t_j	$f(t_{j-1}, y_{j-1})$	$y_j = y_{j-1} + hf(t_{j-1}, y_{j-1})$	$y(t_j)$	$y_j - y(t_j)$
0	0.0	NA	(初始条件)1.0000	1.0000	0
1	0.2	0 − (2)(1) = −2.000	1.0+(0.2)(−2.0)=0.6000	0.6879	−0.0879
2	0.4	0.2 − (2)(0.6) = −1.000	0.6+(0.2)(−1.0)=0.4000	0.5117	−0.1117
3	0.6	0.4 − (2)(0.4) = −0.400	0.4+(0.2)(−0.4)=0.3200	0.4265	−0.1065

使用欧拉法得出的逼近解跟精确解的变化趋势基本一致。任何一步引入的误差都受前一步产生的误差影响。正如步骤$t_3 = 0.6$所示的情况，实际上误差大小可能会一步步地减小。

12.2.1　欧拉法的实现

已知微分方程$dy/dt = f(t, y)$，我们可以将右端的函数按照方程（12-10）编成代码，并使用循环反复求值。例如，方程（12-10）和方程（12-11）可以组合如下：

```
y(1) = ...  % initial condition
for j=2:n
    y(j) = y(j-1) + h * (t(j-1) - 2*y(j-1));
end
```

我们知道，对初始条件t(1)=t，相应地有j = 1。显然，此代码只适用于方程（12-1）的数值积分。因为欧拉法适用于任何一阶ODE，所以开发一个包含欧拉法逻辑的通用程序，

而不是针对特别的ODE的代码是很有用的。这种通用程序需要应用某些计算ODE右端表达式的方法，这可以由feval函数间接地调用右端函数轻易实现（见3.6.3节）。

程序清单12-1中的odeEuler函数是欧拉法的一般实现。其第一个输入参数是diffeq，它是一个存储m文件名字的字符串，该m文件用来计算任何t和y的ODE右端表达式。语句

```
y(j) = y(j-1) + h*feval(diffeq, t(j-1), y(j-1));
```

等价于方程（12-10），只是用feval (diffeq, t (j-1), y (j-1)) 代替了$f(t_{j-1}, y_{j-1})$。

程序清单12-1 函数odeEuler使用简易欧拉法对一阶ODE积分

```
function [t,y] = odeEuler(diffeq,tn,h,y0)
% odeEuler  Euler's method for integration of a single, first order ODE
%
% Synopsis:   [t,y] = odeEuler(diffeq,tn,h,y0)
%
% Input:      diffeq = (string) name of the m-file that evaluates the right
%                      hand side of the ODE written in standard form
%             tn  = stopping value of the independent variable
%             h   = stepsize for advancing the independent variable
%             y0  = initial condition for the dependent variable
%
% Output:     t = vector of independent variable values:  t(j) = (j-1)*h
%             y = vector of numerical solution values at the t(j)

t = (0:h:tn)';            %  Column vector of elements with spacing h
n = length(t);            %  Number of elements in the t vector
y = y0*ones(n,1);         %  Preallocate y for speed

%  Begin Euler scheme; j=1 for initial condition
for j=2:n
   y(j) = y(j-1) + h*feval(diffeq,t(j-1),y(j-1));
end
```

使用odeEuler函数计算特定的ODE时，只需要再创建一个短m文件或内嵌函数对象来计算微分方程的右端表达式。例如，程序清单12-2下半部分的rhs1函数就计算出了方程（12-11）的右端表达式。已知odeEuler.m和rhs1.m文件，前例中手工计算得到的结果可使用下列语句再次得到：

683
≀
684

```
>> [x,y] = odeEuler('rhs1',0.6,0.2,1);
>> disp([x,y])
```

自己试一下！

用下列语句也可以得到相同的结果，它将dy/dt的计算定义为一个内嵌函数对象：

```
>> dydt = inline('t - 2*y','t','y')
>> [x,y] = odeEuler(dydt,0.6,0.2,1);
>> disp([x,y])
```

dydt对象的输入变量必须明确地指定（即dydt = inline ('t - 2*y', 't', 'y')，而非dydt = inline ('t - 2*y')），以避免feval函数在odeEuler中执行时产生运行时错误。明确指定输入变量可使feval既知道输入变量的数目，也知道它们的顺序。

程序清单12-2 函数demoEuler将用欧拉法求得的数值解与方程（12-11）的精确解相比较，函数rhs1用来计算ODE的右端表达式

```
function demoEuler(h)
% demoEuler  Integrate dy/dt = t - 2*y;  y(0) = 1 with Euler's method
%
% Synopsis:  demoEuler(h)
%
% Input:     h  = (optional) stepsize, Default:  h = 0.2
%
% Output:    A table comparing the numerical and exact solutions

if nargin<1,  h = 0.2;  end

tn = 1;  y0 = 1;                        %  stopping time and IC
[t,y] = odeEuler('rhs1',tn,h,1);        %  Euler integration
yex = ( 2*t - 1 + 5*exp(-2*t) )/4;      %  Exact solution

fprintf('      t       y_Euler     y_exact      error\n');
for k=1:length(t)
  fprintf('%9.4f   %9.6f   %9.6f   %10.2e\n',t(k),y(k),yex(k),y(k)-yex(k))
end
fprintf('\nMax error = %10.2e for h = %f\n',norm(y-yex,inf),h);

function dydt = rhs1(t,y)
% rhs1   Evaluate right hand side of dy/dt = t - 2*y
dydt = t - 2*y;
```

将方程（12-7）和方程（12-8）比较，可见欧拉法忽略了系数阶数大于等于h^2的项，所忽略的项就产生了此方法中所谓的离散误差[⊖]。当h减小时，每一步的离散误差也会减小。下一节将详细分析这种影响，在此之前，先通过实验初步研究一下步长对欧拉法精度的影响。

685

例12.4 减小步长的作用

程序清单12-2中的demoEuler函数将欧拉法产生的解和方程（12-11）的精确解作了比较。下面是在$h=0.2$和$h=0.1$时运行demoEuler的结果（为节省篇幅，$h=0.1$情况下的一些步骤没有列出来）。

```
>> demoEuler
    t       y_Euler    y_exact     error
 0.0000   1.000000   1.000000    0.00e+00
 0.2000   0.600000   0.687900   -8.79e-02
 0.4000   0.400000   0.511661   -1.12e-01
 0.6000   0.320000   0.426493   -1.06e-01
 0.8000   0.312000   0.402371   -9.04e-02
 1.0000   0.347200   0.419169   -7.20e-02

Max error =  1.12e-01 for h = 0.200000

 >> demoEuler(0.1);
    t       y_Euler    y_exact     error
 0.0000   1.000000   1.000000    0.00e+00
 0.1000   0.800000   0.823413   -2.34e-02
 0.2000   0.650000   0.687900   -3.79e-02
```

⊖ 也可使用术语截断误差，虽然这两个术语在本文中意思稍有不同。

```
0.3000  0.540000  0.586015  -4.60e-02
 ...
0.8000  0.359715  0.402371  -4.27e-02
0.9000  0.367772  0.406624  -3.89e-02
1.0000  0.384218  0.419169  -3.50e-02

Max error =  5.02e-02 for h = 0.100000
```

这些结果验证了减小h可减小误差的预测。在$h = 0.2$和$h = 0.1$下解的最大误差之比为：

$$\frac{\max(y_i - y(t_i))_{h=0.1}}{\max(y_i - y(t_i))_{h=0.2}} = \frac{0.502}{1.12} = 0.45 \approx \frac{1}{2}$$

将h减小一半就可以将累积误差（粗略地）减小一半，这与理论分析一致：累积误差与h成比例。

图12-4描述了在更大范围的t和三个步长（$h = 0.2$、$h = 0.1$和$h = 0.05$）上，用欧拉法求解方程（12-11）的结果。减小h就可以使每步的误差更小。

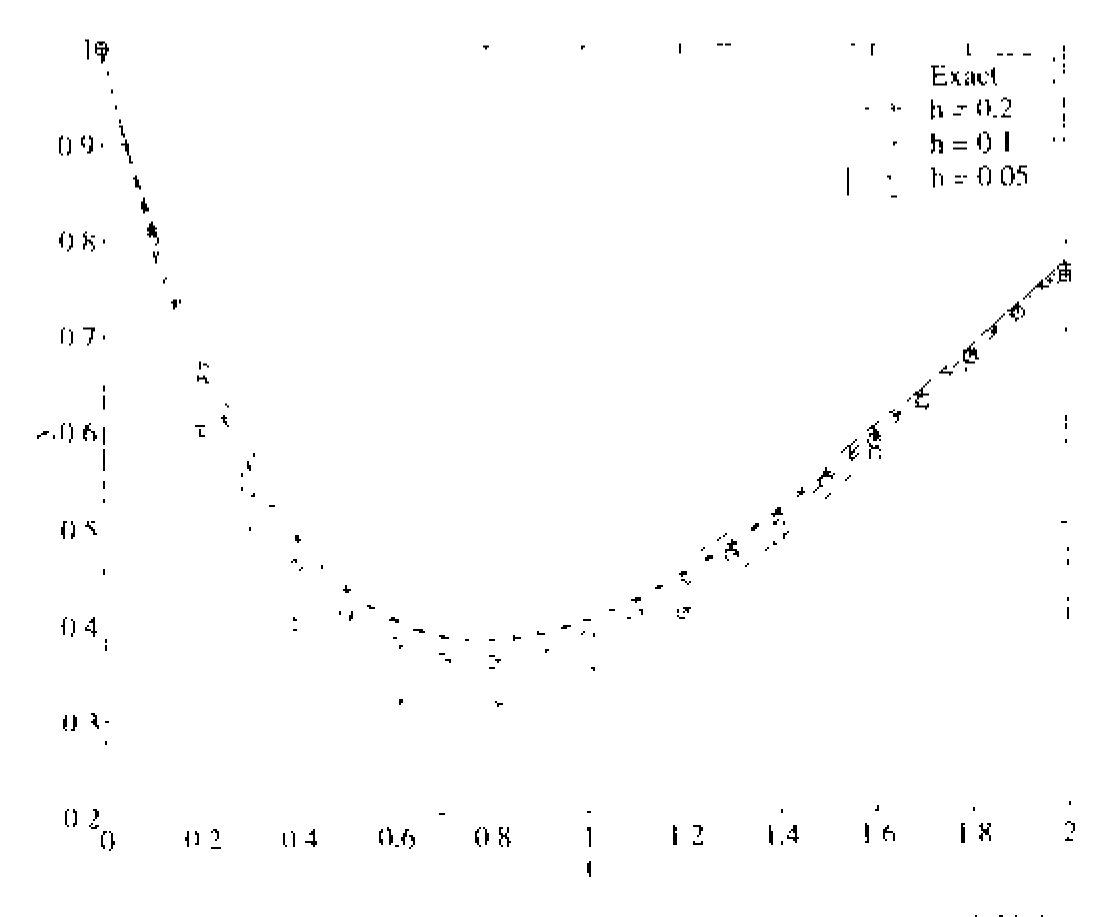


图12-4 在步长$h = 0.2$、$h = 0.1$和$h = 0.05$下，方程（12-11）的精确解与使用欧拉法得到的数值解的比较

686

12.2.2 欧拉法的分析

前面的例子表明了使用欧拉法时误差随h的减小而减小，这个结果在直观上也是有道理的，因为当$h = t_j - t_{j-1}$增大时，用(t_{j-1}, y_{j-1})处的斜率预测$y(t)$将来变化的精度会变差。同样地，预测的精度会依赖于方程（12-2）右端的$f(t, y)$。如果$f(t, y)$变化很慢，或更精确地说，$|f'(t,y)| - |y''(t)|$很小，那么就认为在(t_{j-1}, y_{j-1})的更大邻域内$f(t_{j-1}, y_{j-1})$对$f(t, y)$的估计比$|y''(t)|$较大的情况下更可靠。本节将这些直观思想进行了量化。在初值问题中，除非知道精确解，否则不可能计算出数值解误差的精确大小。现在分析的目的不是要精确地测量误差，而是要确定误差的数量级随h的减小而下降得有多快。

评估方程（12-2）数值解的逼近程度有两种观点。可以考虑任意一步中产生的误差大小，或考虑在得出解的整个区间$t_0 \leq t \leq t_N$上的误差。两种观点在本章中都要用到。其中，任意一步的误差称为局部离散误差（local discretization error, LDE），整个区间的误差称为全局离散误

[687] 差（global discretization error,GDE）。之所以使用术语离散误差来代替截断误差，是因为它更能表达出误差的理论表达式是如何得到的。局部离散误差的表达式不关心泰勒级数展开式中的丢弃的项，而是通过比较原始ODE的离散数值解所满足的方程得到。

局部离散误差　欧拉法的计算公式（方程（12-10））可以重新整理成

$$\frac{y_j - y_{j-1}}{h} = f(t_{j-1}, y_{j-1}) \tag{12-12}$$

此式是对方程（12-2）中的原始ODE的离散化模拟。左端是导数dy/dt的逼近。由欧拉法产生的数值解就是满足所谓的微分方程（12-12）的离散函数y_j。

精确解和逼近解并不相同，但是这并不奇怪，因为它们由两个不同的方程（分别为方程（12-2）和方程（12-12））决定。实际上，如果在每一步都将精确解代入微分方程，那么方程（12-12）中的相等关系就不复存在，即

$$\frac{y(t_j) - y(t_{j-1})}{h} - f(t_{j-1}, y(t_{j-1})) \neq 0 \tag{12-13}$$

局部离散误差就是以上方程左端距离零远近的一个度量。

使用方程（12-13）来度量局部离散误差的一个技术问题是，除非$j = 0$，精确解不可能每一步都与(t_{j-1}, y_{j-1})一致。为校正这个问题，引进一个辅助函数组$z_j(t)$，并假设它们是初值问题的精确解（比较$z_j(t)$和方程（12-2））：

$$\frac{\mathrm{d}z_j}{\mathrm{d}t} = f(t, z_j), \quad z_j(t_{j-1}) = y_{j-1}, \quad j = 1, \ldots, n-1 \tag{12-14}$$

$z_j(t)$是在第j个子区间满足方程（12-14）并且穿过上一个数值解的点(t_{j-1}, y_{j-1})的连续函数。方程（12-11）的$y(t)$、$y_j(t_j)$和$z_j(t)$之间关系如图12-5所示。

现在将$z_j(t)$代换到方程（12-12）中，就可以精确地定义局部离散误差$\tau(t, h)$了：

$$\tau(t, h) = \frac{z_j(t_j) - z_j(t_{j-1})}{h} - f(t_{j-1}, z_j(t_{j-1})) \tag{12-15}$$

局部离散误差（LDE）就是由方程（12-14）的精确解来计算微分方程时得到的残差。记号$\tau = \tau(t, h)$暗示了离散误差决定于h和t，其中h的值由用户控制，t是某个区间内的位置，此区间决定了函数$z_j(t)$（或$y(t)$）的导数大小。

[688]

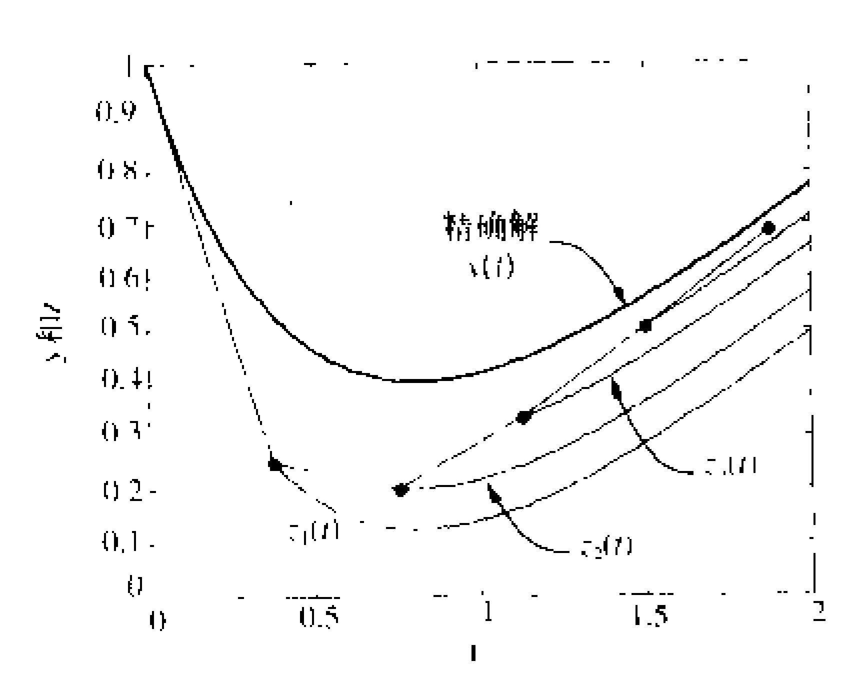


图12-5　在$dy/dt = t - 2y$，$y(0) = 1$和$h = 0.4$时，欧拉法得到的数值解（实心点）与精确解$y(t)$。$z_j(t)$是$dz/dt = t - 2z_j$，$z_j(t_{j-1}) = y_{j-1}$的精确解

方程（12-15）不是最有用的形式，因为它没有清楚地表明离散误差对h的依赖关系。再进行一下小小的改进，就可以得到明确的依赖关系。在点(t_{j-1}, y_{j-1})处用泰勒级数扩展$z_j(t)$，得

$$z_j(t_j) = z_j(t_{j-1}) + hz_j'(t_{j-1}) + \frac{h^2}{2} z_j''(\xi) \tag{12-16}$$

其中ξ是在闭区间$[t_{j-1}, t_j]$上的某个点。以上方程可以重新整理为

$$\frac{z_j(t_j) - z_j(t_{j-1})}{h} = z_j'(t_{j-1}) + \frac{h}{2} z_j''(\xi) \tag{12-17}$$

将方程（12-14）中ODE的右端代入方程（12-17）并重新整理，得

$$\frac{z_j(t_j) - z_j(t_{j-1})}{h} - f(t_{j-1}, z_{j-1}) = \frac{h}{2} z_j''(\xi)$$

将此式与方程（12-15）比较可知，第j个子区间中欧拉法的LDE为

$$\tau(t_j, h) = \frac{h}{2} z_j''(\xi)$$

LDE受h和$z_j''(\xi)$影响，其中h由用户控制，$z_j''(\xi)$由被积ODE决定。我们只知道ξ在第j个子区间中，因此$z_j''(\xi)$的值计算不出来。这并非是多么严重的问题。假设$z_j''(\xi)$在整个区间$t_0 \leqslant t \leqslant t_N$上都不超过值$M$。那么对每一步，都有

689

$$\tau(t, h) \leqslant \frac{hM}{2} \tag{12-18}$$

此方程提供了局部离散误差的上限。在预测h的改变对数值解的精度有什么影响时，这个方程就是基础。虽然$\tau(t, h)$的大小未知，但方程（12-18）表明了局部离散误差会与h成比例地减小。如果两个解是由不同h值的欧拉法得到，那么解的离散误差的比率为：

$$\boxed{\frac{\tau(t, h_2)}{\tau(t, h_1)} = \frac{h_2}{h_1}\text{，欧拉法中}h\text{对LDE的影响}} \tag{12-19}$$

此比率与M无关，M是由ODE及其初始值决定的。注意，虽然方程（12-15）适用于一般情况，但是方程（12-18）和方程（12-19）却只适用于欧拉法。一般地，其他方法的局部离散误差将是h^p的函数，其中p是方法所谓的阶数。

方程（12-18）在估计本章各种数值方法的局部离散误差时非常典型。比较各种方法，我们感兴趣的主要是精度对h的依赖关系。因此，离散误差（局部的和全局的）使用阶符来表示（另见5.3.2节）。方程（12-18）表明欧拉法有一个$O(h)$的局部离散误差。既然对欧拉法来说$p = 1$，那么就可以说这个方法具有一阶精度。注意离散误差的阶与应用此方法的ODE的阶没有任何关系。

在本书的前几章，我们主要着重于单个数值方法的截断误差。比较方程（12-7）和方程（12-8），我们发现欧拉法的截断误差是$O(h^2)$。这是一个采用截断误差还是离散误差来分析ODE的问题。离散误差的优点是局部和全局离散误差（见下节）对单步法来说有相同的数量级。

全局离散误差 LDE表明了在每一步计算出的数值解偏离最近的真实解的程度。虽然这个信息很有用，但是知道在整个积分区间上最大误差如何依赖于h会更有用。全局离散误差（GDE）是区间上的最大误差，通过对GDE的分析，可以得出整个区间上误差对h的依赖关系。幸运的是，局部离散误差和全局离散误差之间还有直接的联系。

690

为定义GDE，首先需要定义如何来度量连续函数和离散函数的差。因为y_j只在点t_j上有定

义，其中$j=0$，…，N，那么也只有这些点上的误差才有意义。令e_j为所求解上的离散误差（即$e_j=y_j-y(t_j)$）那么，一般地，GDE为

$$\max(|e_j|)=\max(|y_j-y(t_j)|) \qquad j=1,\cdots,N \tag{12-20}$$

（为什么忽略$j=0$？）

对方程（12-2）中的一般初值问题，欧拉法的GDE是

$$\max|y(t_j)-y_j|\leqslant \frac{h}{2}\frac{M}{L}\left[e^{L} -1\right]=O(h) \tag{12-21}$$

其中M和L是在$t_0\leqslant t\leqslant t_N$上依赖于$f(t,y)$的常数（见[9、31、65、70]中关于方程（12-21）的证明）。常数M和L不是主要关心的对象，因为它们是由$y(t)$的数学性质决定的。但并不是说M和L就一点都不重要了。一般情况下，既然$y(t)$未知，那么就不可能计算出M和L来。

由方程（12-21）得到的最重要的信息是欧拉法的全局离散误差与h之间是线性关系，此结论解释了例12.4中数值实验的结果。

12.2.3 一般化：单步法

欧拉法在实际中很少用到，因为还有其他精度更高（更小的LDE和GDE）且不太难实现的方法。欧拉法是一类所谓的单步法中最简单的方法。单步法使用如下形式的方程来改进ODE的解：

$$y_j=y_{j-1}+h\Phi(t,y,h,f) \tag{12-22}$$

其中$\Phi(t,y,h,f)$由方程（12-2）中ODE的右端得到。对欧拉法：

$$\Phi(t,y,h,f)=f(t_{j-1},y_{j-1}) \tag{12-23}$$

欧拉法中的Φ函数与h和f无关，其他单步法中的Φ函数依赖于t、y、h和f。

在单步法中，y_j值只依赖于前一步得到的数值（即y_j依赖于y_{j-1}和t_{j-1}）。本节中介绍的单步法涉及$f(t,y)$在$t_{j-1}\leqslant t\leqslant t_j$中$t$的中间值上多次求值，这样做的优点是离散误差会由于每一步作更多计算而减小。需要记住，虽然$f(t,y)$在区间$t_{j-1}\leqslant t\leqslant t_j$中的多个点上求值，但是此法仍然是单步法，因为只有前一步的解（即y_{j-1}）才出现在方程（12-22）的右端。

691

单步法的离散误差可以使用12.2.2节中介绍的方法进行分析。对方程（12-2）的初值问题，任何单步法的LDE为（参考方程（12-15））

$$\tau(t,h)\leqslant Ch^{p}=O(h^{p}) \tag{12-24}$$

其中$p\geqslant 1$，其GDE为

$$\max|y(t_j)-y_j|=O(h^{p}) \tag{12-25}$$

（例见[9、31、65、70]中有关的证明）。换言之，单步法的LDE为$O(h^p)$，GDE也为$O(h^p)$。

12.2.4 本节小结

欧拉法用来阐明ODE数值方法的重要特性。这些特性有：

- 数值积分法由截断逼近ODE的泰勒级数推导出来。
- 实现时将ODE积分法与右端$f(t,y)$的计算分开。通用的ODE计算程序要求用户提供一个小m文件来计算$f(t,y)$。

- 局部离散误差（LDE）是每一步的误差。一般地，对单步法来说，LDE的形式为

$$\text{LDE} = O(h^p)$$

其中h为步长；p是一个整数，且有$p \geqslant 1$。

- 将一个区间划分成步长为h的几（个计算）步，在每一步上应用ODE积分法时，全局离散误差（GDE）是LDE的累积结果。使用本书中LDE的定义法，GDE为

$$\text{GDE} = O(h^p)$$

12.3 高阶单步法

在前面的部分中，欧拉法用来阐述很多基本思想——特别是它阐明了单步法的特性。在本节中，将逐渐讨论一系列精度更高的单步法。这里论述速度会更快，每个方法可以看成是欧拉法思想的提高。各种算法的介绍将按照精度由低到高（GDE由大到小）依次进行。 [692]

12.3.1 中点法

增加欧拉法精度的惟一方法就是减小h。这样，要么沿t轴会计算更多的数值解，要么在每一步中多次计算斜率，以减小LDE。中点法在子区间的中点使用实验性的Euler步骤，这样就要重新计算整个区间的斜率。

图12-6描述了中点法的其中一步。假设因变量的值y_{i-1}在t_{i-1}处已知。定义k_1为点（t_{i-1}，y_{i-1}）处所计算的斜率：

$$k_1 = f(t_{i-1}, y_{i-1})$$

然后使用k_1来估计区间中点处的解，有：

$$y_{i-1/2} = y_{i-1} + \frac{h}{2} f(t_{i-1}, y_{i-1})$$

换言之，$y_{i-1/2}$应该是由欧拉法想要得到的y在$t_{i-1}+h/2$的值。接下来，估计中点处的斜率，称其为k_2：

$$k_2 = f\left(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2} k_1\right)$$

最后，计算出y在整个区间末端的值

$$y_i = y_{i-1} + hk_2$$

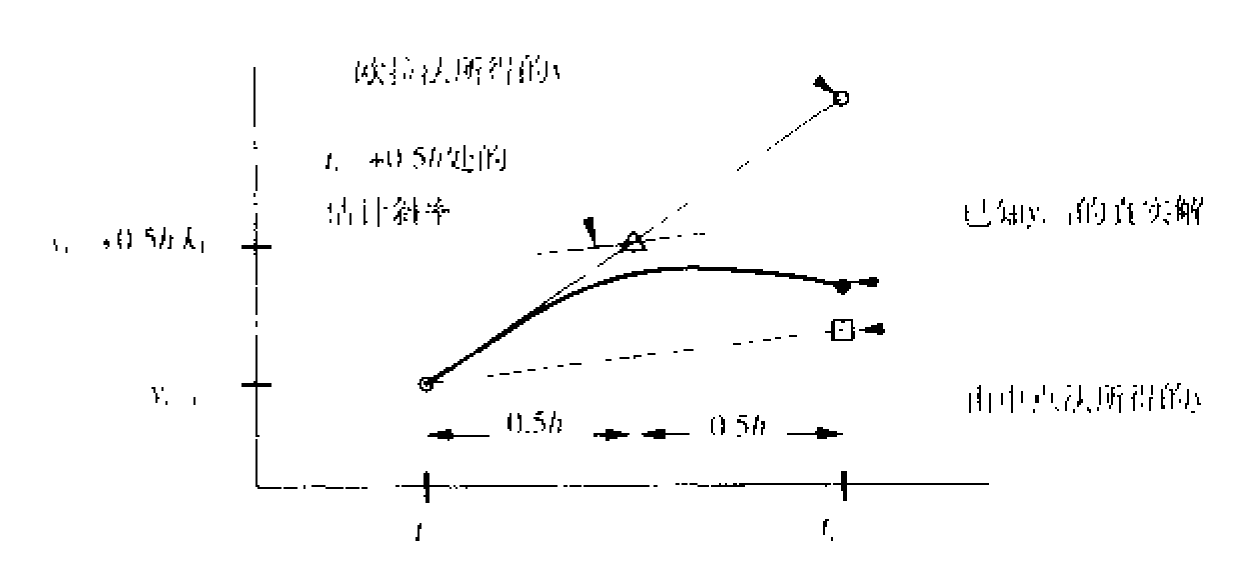


图12-6 中点法的图形化描述 [693]

中点法的局部和全局离散误差均为$O(h^2)$。精度的增加是通过在每一步将$f(t, y)$的计算次数加倍来实现的。这是否值得呢？答案是值得的，下面的例12.5将会加以证明。

只需要在欧拉法的实现代码上作很小的改变，就可以实现中点法，结果就是程序清单12-3中的odeMidpt函数。

程序清单12-3 函数odeMidpt使用中点法对一阶ODE积分

```
function [t,y] = odeMidpt(diffeq,tn,h,y0)
% odeMidpt  Midpoint method for integration of a single, first order ODE
%
% Synopsis: [t,y] = odeMidpt(diffeq,tn,h,y0)
%
% Input:    diffeq = (string) name of the m-file that evaluates the right
%                    hand side of the ODE written in standard form
%           tn  = stopping value of the independent variable
%           h   = stepsize for advancing the independent variable
%           y0  = initial condition for the dependent variable
%
% Output:   t = vector of independent variable values:  t(j) = (j-1)*h
%           y = vector of numerical solution values at the t(j)

t = (0:h:tn)';            % Column vector of elements with spacing h
n = length(t);            % Number of elements in the t vector
y = y0*ones(n,1);         % Preallocate y for speed
h2 = h/2;                 % Avoid repeated evaluation of this constant

%  Begin Midpoint scheme; j=1 for initial condition
for j=2:n
   k1 = feval(diffeq,t(j-1),y(j-1));
   k2 = feval(diffeq,t(j-1)+h2,y(j-1)+h2*k1);
   y(j) = y(j-1) + h*k2;
end
```

例12.5 中点法与欧拉法比较

考虑选择中点法或欧拉法对ODE积分。对给定的h，中点法明显地更精确——其全局离散误差为$O(h^2)$，而欧拉法的全局离散误差为$O(h)$。另外，中点法对$f(t, y)$程序的调用次数也是欧拉法的两倍。假设求解所用的h越来越小，并且令一个解的h是前一个解所用h的一半。连续运行可知，中点法的GDE减小为原来的1/4，而欧拉法的GDE则减小为原来的1/2。

[694]

程序清单12-4 函数compEM比较对方程（12-26）使用中点法和欧拉法得到的数值解

```
function compEM
% compEM     Compare Euler and Midpoint for solution of dy/dt = -y;  y(0) = 1
%
% Synopsis:  compEM
%
% Input:     none
%
% Output:    Flops and global trunc errors for a sequence of stepsizes

tn = 1;  y0 = 1;        % Length of interval and initial condition
```

```
fprintf('\n    h        flopsE    errE      flopsM    errM\n');
for h = [0.2  0.1  0.05  0.025  0.0125  0.00625]
  flops(0);  [te,ye] = odeEuler('rhs2',tn,h,1);  flopse = flops;  %  Euler
  flops(0);  [tm,ym] = odeMidpt('rhs2',tn,h,1);  flopsm = flops;  %  Midpoint
  % --- global discretization errors
  yex = y0*exp(-te);          %  Exact solution at discrete t
  erre = max(abs(ye-yex));   errm = max(abs(ym-yex));
  fprintf('%8.5f %7d %11.2e %7d %11.2e\n',h,flopse,erre,flopsm,errm);
end
```

要将减小的离散误差量化，可使用中点法和欧拉法来解

$$\frac{dy}{dt} = -y, \quad y(0)=1, \quad 0 \leqslant t \leqslant 1 \tag{12-26}$$

然后将数值解与精确解$y = e^{-t}$比较。

程序清单12-4中的compEM函数使用逐渐减小的h求方程（12-26）的数值解，计算工作量使用内置的flops函数度量。对每个h，打印出两种方法的浮点操作次数和全局离散误差。运行compEM得到如下结果：

```
>> compEM
 h          flopsE    errE        flopsM    errM
0.20000        31     4.02e-02       57     2.86e-03
0.10000        61     1.92e-02      112     6.62e-04
0.05000       121     9.39e-03      222     1.59e-04
0.02500       241     4.65e-03      442     3.90e-05
0.01250       481     2.31e-03      882     9.67e-06
0.00625       961     1.15e-03     1762     2.41e-06
```

695

果然，对给定的h，欧拉法耗费的工作量大约为中点法的一半，但所得解的精度也明显降低。对方程（12-26），欧拉法需要$h = 0.0125$才能得到与$h = 0.2$的中点法精度相当的解。要得到相同的精度，欧拉法需要多于中点法八倍的工作量（481次浮点操作对57次浮点操作）。明显地，中点法是一种更高级的技术。

中点法在精度方面的改进与其$O(h^2)$的形式精度（formal accuracy）是一致的。例如，将步长从0.025减小到0.0125会将误差减小为原来的1/4：

$$\frac{\max(|e_i|)_{h=0.025}}{\max(|e_i|)_{h=0.0125}} = \frac{3.90e-0.5}{9.67e-06} = 4.03 \approx 2^2$$

12.3.2 Heun法

Heun法，也叫改进欧拉法，它与中点法类似，也在每一步中计算两次斜率。Heun法可形象化地描述为图12-7。像中点法一样，它也定义k_1为点（t_{j-1}，y_{j-1}）处计算的斜率：

$$k_1 = f(t_{j-1}, y_{j-1})$$

接着，使用k_1来估计区间末端的斜率k_2：

$$k_2 = f(t_{j-1}+h, y_{j-1}+hk_1)$$

它的值与中点法中k_2的值不同。最后，使用斜率k_1和k_2的平均值计算y_j：

$$y_j = y_{j-1} + h\frac{k_1 + k_2}{2}$$

欧拉法所得的y_i

t_i处估计的斜率

Heun法所得的y_i

给定y_{i-1}的真实解

y_{i-1}

h

t_{i-1}　　t_i

696

图12-7　Heun法的图形化描述

Heun法的局部和全局离散误差是$O(h^2)$，它在步长为h的每一步需要计算两个函数。因此，Heun法的形式精度及计算工作量都与中点法的相等。练习10的目标就是要实现Heun法。

12.3.3　四阶Runge-Kutta法

Heun法使用区间$t_{i-1} \le t \le t_i$上的两个斜率的平均值来逼近y_i的值。那么使用更多斜率的平均值是否可以得到更好的精度（即更低的离散误差）呢？确实如此，但是更多斜率的简单平均仍然比不上加权平均。由多个斜率计算y_i的一般公式为：

$$y_i = y_{i-1} + \sum_{j=1}^{m} \gamma_j k_j$$

其中γ_ℓ是权系数，k_ℓ是第j个子区间中不同点处计算的斜率。对Heun法来说，有$\gamma_1=\gamma_2$。一般地，权要满足

$$\sum_{j=1}^{m} \gamma_j = 1$$

有一个形式化过程用来求出γ_j和k_j，使最后的逼近值在所期望的离散误差之内，此方法就称为Runge-Kutta法，该方法是由两个独立推导出这个方法的人物而得名。

包含这种思想的一个非常通用和易懂的方法就是四阶Runge-Kutta法，这里我们简称为"RK-4"。Runge-Kutta法的阶与其GDE的阶相等。对RK-4，局部和全局离散误差为$O(h^4)$。在RK-4法中步长为h的每一步，对斜率要估计四次，如图12-8所示。这些斜率估计值k_1、k_2，k_3和k_4的计算如下：

697

$$k_1 = f(t_{i-1}, y_{i-1})$$

$$k_2 = f\left(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}k_1\right)$$

$$k_3 = f\left(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{h}{2}k_2\right)$$

$$k_4 = f(t_{i-1} + h, y_{i-1} + hk_3)$$

使用k_i的加权平均值来计算下一个y值：

$$y_j = y_{j-1} + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right) \quad (12\text{-}27)$$

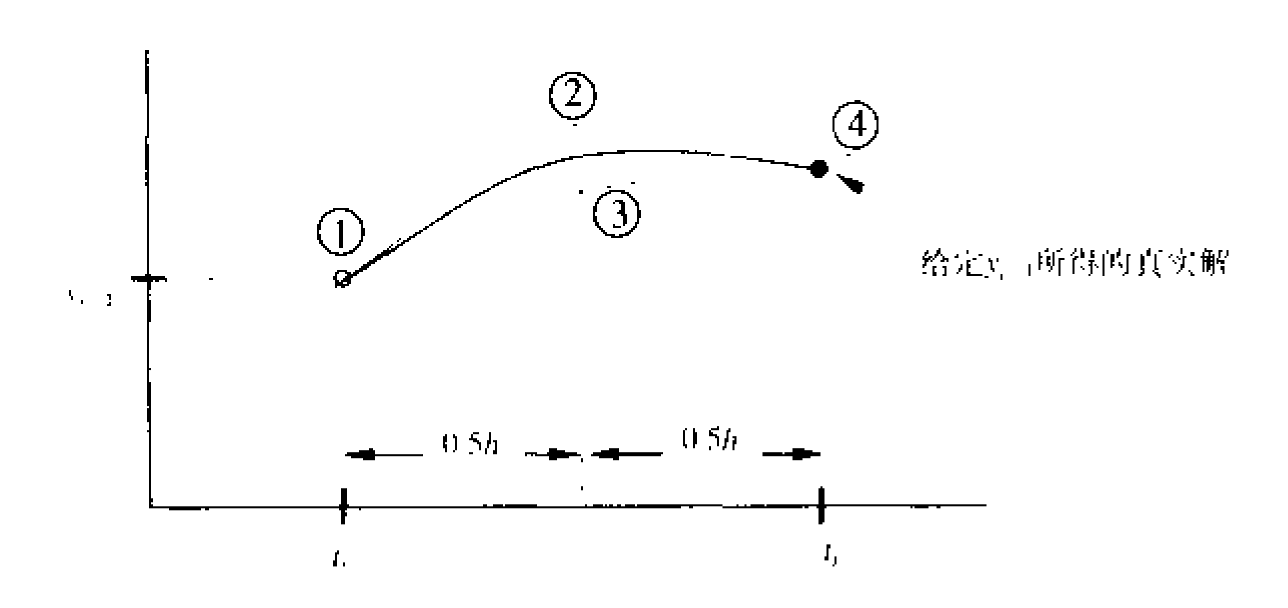


图12-8　四阶Runge-Kutta法中计算斜率的四个点的图形化描述

程序清单12-5中的odeRK4函数使用与odeEuler和odeMidpt函数相同的代码结构来实现RK-4法。其中主要的不同就是每一步额外斜率值的计算。RK-4法中每一步需要的计算工作量是欧拉法的四倍，中点法的两倍。虽然每一步的计算工作量变大了，但是离散误差值的减小使RK-4法成为实现期望精度结果的一个更有效的方法。

程序清单12-5　函数odeRK4使用四阶Runge-Kutta法来对一阶ODE积分

```
function [t,y] = odeRK4(diffeq,tn,h,y0)
% odeRK4  Fourth order Runge-Kutta method for a single, first order ODE
%
% Synopsis:  [t,y] = odeRK4(fun,tn,h,y0)
%
% Input:     diffeq = (string) name of the m-file that evaluates the right
%                     hand side of the ODE written in standard form
%            tn  = stopping value of the independent variable
%            h   = stepsize for advancing the independent variable
%            y0  = initial condition for the dependent variable
%
% Output:    t = vector of independent variable values:  t(j) = (j-1)*h
%            y = vector of numerical solution values at the t(j)

t = (0:h:tn)';                     % Column vector of elements with spacing h
n = length(t);                     % Number of elements in the t vector
y = y0*ones(n,1);                  % Preallocate y for speed
h2 = h/2;  h3 = h/3;  h6 = h/6;    % Avoid repeated evaluation of constants

% Begin RK4 integration; j=1 for initial condition
for j=2:n
   k1 = feval(diffeq, t(j-1),    y(j-1)        );
   k2 = feval(diffeq, t(j-1)+h2, y(j-1)+h2*k1  );
   k3 = feval(diffeq, t(j-1)+h2, y(j-1)+h2*k2  );
   k4 = feval(diffeq, t(j-1)+h,  y(j-1)+h*k3   );
   y(j) = y(j-1) + h6*(k1+k4) + h3*(k2+k3);
end
```

例12.6 欧拉法、中点法和RK-4法的比较

本例延续例12.5中的比较，再包括与RK-4法的比较。函数compEMRK4（此处未列出，包含在NMM工具箱内）中的代码度量出了在逐渐减小的h上用欧拉法、中点法和RK-4法来解方程（12-26）时的浮点操作次数和全局离散误差。

```
>> compEMRK4
  h         flopsE    errE        flopsM    errM        flops4    err4
 0.20000      31     4.02e-02      57      2.86e-03      129     5.80e-06
 0.10000      61     1.92e-02     112      6.62e-04      254     3.33e-07
 0.05000     121     9.39e-03     222      1.59e-04      504     2.00e-08
 0.02500     241     4.65e-03     442      3.90e-05     1004     1.22e-09
 0.01250     481     2.31e-03     882      9.67e-06     2004     7.56e-11
 0.00625     961     1.15e-03    1762      2.41e-06     4004     4.70e-12
```

正如对离散误差估计的那样，RK-4法的误差比另两种方法的误差小得多。虽然该方法可以明显地减小离散误差，但使用浮点操作次数来度量时，RK-4法的工作量要比另外两种方法大。

699

下表重组了compEMRK4的运行结果，以便比较每种方法得到大略相等的精度需要的工作量：

	误差	h	浮点操作次数
欧拉法	1.2×10^{-3}	0.00625	961
中点法	6.6×10^{-4}	0.1	112
中点法	2.4×10^{-6}	0.00625	1762
RK-4	3.3×10^{-7}	0.1	254

要得到相当的精度（误差为$O(10^{-3})$），中点法需要的浮点操作次数大约为欧拉法的1/8。RK-4法要达到与中点法相当的精度（误差为$O(10^{-6})$），需要的浮点操作次数大约为中点法的1/7。由此我们推断中点法比欧拉法有效得多，而RK-4法又比中点法和欧拉法有效得多。比较的结果严格地适用于方程（12-26）中的问题模型，但是其他的ODE也可以得到类似的结果。

最后，compEMRK4的输出可以用来验证RK-4法的理论误差估计。在$t = 1.0$处，采用$h = 0.00625$和$h = 0.0125$的RK-4法得到的误差比为

$$\frac{\max(|e_j|)_{h=0.00625}}{\max(|e_j|)_{h=0.0125}} = \frac{4.7\times10^{-12}}{7.56\times10^{-11}} = 0.0622 \approx (0.499)^4$$

因此，h减小一半会使最大误差减小到（大约）二分之一的4次方（即减小为原来的1/16）。这与RK-4的全局离散误差是$O(h^4)$相一致。

12.4 自适应步长算法

根据前面的描述，有两种增加ODE数值解精度的方法：减小h和选择更精确的算法。由欧拉法到RK-4法通过增加阶数取得了巨大的成功，但不幸的是这种成功却没能在高于四阶的Runge-Kutta法上得到继续。精度可以提高，但是同时要大大增加ODE右端表达式的计算工作，这个巨大代价使这些高阶算法失去了吸引力。提高效率的关键是选择合适的算法。特别地，求数值解所需要的总步数通常可以由自适应改变的h来减少。虽然迄今为止我们都是使用固定的h，但是为什么不能在求数值解过程中的每一步改变h呢？这里惟一复杂的因素就是设计一个准则来自动选择h。

为提高效率，调整h的算法必须将h的改变与解的局部精度相关联。调整的目的是在需要满足用户自定义容差时减小h，在不会导致误差超过用户自定义容差时增大h。要使其顺利进行，ODE积分算法必须用一种方法来测量求解过程中产生的误差。这看起来似乎不可能，因为精确解不知道，否则我们就不需要数值解了。然而可以证明，有关LDE的估计足够用来控制h的大小。 700

估计LDE的一个方法是使用两个不同的h计算逼近解并比较结果。如果由一个大h值和一个小h值得到的解相同，那么减小h就没有意义了；相反，如果这两个解明显不同，那么我们可以假设由大h值得到的解是不精确的。显然，这个过程需要在每一步都计算两个数值解。

另一种估计LDE的方法是使用两种h值相同、LDE不同的算法同时求解。如果低精度算法的结果与高精度算法的（接近）相同，那么就没有必要减小h；相反，如果两种算法得到的y_i明显不同，那么就应该减小h。内置程序ode23和ode45就采用了这种方法。

ode23函数和ode45函数

内置的ode23和ode45函数使用称为Runge-Kutta-Fehlberg法的自适应步长算法。Runge-Kutta-Fehlberg法在大小为h的区间内仔细选择子步（substep）集合，并在每一步都同时得到两个有不同离散误差的解。在每一步计算两个解可以监测解的精度，并且可以依照用户自定义容差来调整h的大小。

程序ode23同时使用2阶和3阶Runge-Kutta公式来改进解并且监测精度。类似地，ode45同时使用4阶和5阶公式。Runge-Kutta-Fehlberg算法很优秀，它不仅提供了监测精度的机制，而且两个公式间还可以共享某些中间斜率值（k_1、k_2等等）。例如，ode45程序不需要计算9个（RK-4的4个和RK-5的5个）斜率值，只需要计算6个，这些斜率求解可以简单地实现。见[61、65]中关于Runge-Kutta-Fehlberg法的讨论和高效的代码实现。ode45程序基于Dormand和Prince [18]的算法。 701

ode45函数可以由以下几种方法调用[⊖]：

```
[t,Y] = ode45(diffeq,tn,y0)
[t,Y] = ode45(diffeq,[t0 tn],y0)
[t,Y] = ode45(diffeq,[t0 tn],y0,options)
[t,Y] = ode45(diffeq,[t0 tn],y0,options,arg1,arg2, ...)
```

ode23使用相同的参数序列：第一个参数*diffeq*是一个m文件的名字，该m文件用来计算微分方程的右端表达式；参数*t0*和*tn*定义了求解的总区间，如果二者都已知，其值必须存储在一个向量中，记为[*t0*, *tn*]，如果*t0*未知，就假设它为零；*y0*为初始条件。参数*option*在下面的子部分中讨论。ode45并不直接使用参数*arg1*、*arg2*、...，而是将其传给用户自定义的*diffeq*程序。下面就介绍这些传递参数的用法并用例子示范它们的使用。

与以前介绍的程序不同，ode23和ode45都不需要确定的h值，它们只需要调整h的值，以使

$$\bar{\tau} < \max(\texttt{RelTol}\times|y_i|, \texttt{AbsTol})$$

成立。其中$\bar{\tau}$是对LDE的估计，RelTol和AbsTol是容差（error tolerance，或称容许误差），

⊖ 也可以使用其他形式。在MATLAB提示符下键入help ode45可得到更多信息。

y_j是在第j步计算的y值，容许误差的缺省值为

$$\text{RelTol}=1\times10^{-3},\quad \text{AbsTol}=1\times10^{-6}\text{（缺省值）}$$

子条件中的max()操作表示控制步长调整的容差中的较大者。

例如，假设$|y_j|\sim100$。RelTol和AbsTol的缺省值表示最大可接受的误差估计是$0.1=1\times10^{-3}\times100$。这种情况下，绝对容差（$1\times10^{-6}$）比RelTol$\times y_j$小得多。相对容差$1\times10^{-3}$说明所计算解有0.1%的不确定性。相反，如果$|y_j|\sim10^{-5}$，那么就可以用$1\times10^{-6}$（缺省）的绝对容差，所计算解中的不确定性就有10%。因此，当解很小或y_j的值接近重要的零交点时，AbsTol应该作相应的调整。RelTol和AbsTol的值随ode45的*options*参数的不同而改变。

ode45的各种特征可通过一系列的例子来研究。首先，我们示范ode45最简单的用法。

例12.7　使用ode45解ODE

[702]　下面的语句使用ode45来求解方程（12-26），并比较了所选定t值上的精确解与数值解。

```
>> tn = 1; y0 = 1;                  % stopping time and initial condition
>> [t,y] = ode45('rhs2',tn,y0);     % ode45 solution
>> yex = y0*exp(-t);                % Exact solution
>> norm(y-yex,inf)
ans =
     5.7038e-09
```

所得6×10^{-9}的GDE与定步长为$h=0.025$的RK-4法得到的GDE相当（比较例12.6）。对如此简单的ODE，使用ode45中复杂的计算机制没有太大优势。但是对一般应用，仍然强烈推荐ode45。

用odeset控制ode45　内置ODE积分程序（另见表12-3）的执行由多个参数来控制，这些参数的缺省值给ODE的解提供了一个合理的起始点。函数odeset就是调整这些参数的用户接口。这些控制参数在调用ode45时不能直接改变。

调整控制参数有两个步骤。首先，产生一个包含所要改变的参数值的数据结构[⊖]变量。然后，将此数据结构传给ode45。odeset的一个典型应用类似于：

```
options = odeset('parameterName',parameterValue, ... );
[t,y] = ode45(diffeq,tn,y0,options);
```

*parameterName*是要改变参数的名字，*parameterValue*是赋给参数的数值。odeset的输入列出了*parameterName*和*parameterValue*对，在命令提示符处输入odeset，就可以得到可能的参数。

```
>> odeset
         AbsTol: [positive scalar or vector 1e-6 ]
            BDF: [on | off ]
         Events: [on | off ]
    InitialStep: [positive scalar ]
       Jacobian: [on | off ]
      JConstant: [on | off ]
```

⊖ 数据结构是一种特殊的变量类型，它可以包含其他变量的任意集合。由odeset输出的options变量就是一个数据结构，它包含了字符串、数值对。数据结构是MATLAB第5版以后的高级特性。在提示符下键入help struct可以得到更多的信息。使用了odeset函数就不需要也不应该再去直接改变options数据结构的内容。

703

```
      JPattern: [on | off ]
          Mass: [on | off ]
  MassConstant: [on | off ]
      MaxOrder: [1 | 2 | 3 | 4 | 5 ]
       MaxStep: [positive scalar ]
     OutputFcn: [string ]
     OutputSel: [vector of integers ]
        Refine: [positive integer ]
        RelTol: [positive scalar 1e-3 ]
         Stats: [on | off ]
    Vectorized: [on | off ]
```

并不是所有这些参数都适用于每个ODE求解程序. ODE求解程序选项的值可以单个或成组赋予. 例如, 用

```
>> options = odeset('InitialStep',0.001)
```

可将InitialStep参数置为0.001。用

```
>> options = odeset('InitialStep',0.001,'MaxStep',0.2)
```

可将InitialStep参数置为0.001, MaxStep参数置为0.2

指定options数据结构中的值, 将数据结构作为第四个输入参数传给ode45, 即

```
>> [t,y] = ode45('myODE',tn,y0,options);
```

例12.8和例12.9示范了odeset的用法。要想知道更多信息, 可在MATLAB命令提示行中键入help odeset。

例12.8 用odeset控制ode45的应用

本例介绍了如何使用odeset函数来控制由ode45得到的解. 程序清单12-6中的demoODE45opts函数将使用ode45得到方程 (12-26) 的数值解, 并将数值解与精确解进行比较. 输入值rtol和atol取代了ode45中RelTol和AbsTol的缺省值. 如果没有输入值提供给demoODE45opts函数, 函数就使用RelTol和AbsTol原始的缺省值。

函数odeset也用来改变Refine参数, 此参数控制是否在t轴更小的细分上对ode45的输出进行插值. 使用插值, ode45可以用相当大的h来得到高精度的结果, 因此, 在很多问题中插值还是值得的。ode45使用的h可以非常大, 大到结果图形看上去都不能接受, 但得到的解还是落在用户自定义的容许误差之内. 参数Refine定义了在解 (t_i, y_i) 中间产生的插值数量。对ode45来说, Refine的缺省值是4, 表示ode45返回的数组t和y中的点会是实际上求数值解所用点的四倍. 额外的输出点就在这些数值解之间进行插值, 这样Refine对解的精度就不会有影响. 在demoODE45opts中, Refine的缺省值设为1, 因此, 可以将打印出的结果与例12.5和例12.6中的结果直接比较。

704

程序清单12-6 函数demoODE45opts由用户选择容差参数使用ode45来解方程 (12-26)

```
function demoODE45opts(rtol,atol,nref)
% demoODE45opts  Integrate dy/dx = -y;  y(0) = 1 with ode45 and options
%
% Synopsis:  demoODE45opts
%            demoODE45opts(rtol)
%            demoODE45opts(rtol,atol)
%            demoODE45opts(rtol,atol,nref)
```

```
%
% Input: rtol = (optional) relative tolerance used by ode45
%               Default: rtol = 1e-3 (internal default used by ode45)
%        atol = (optional) absolute tolerance used by ode45
%               Default: atol = 1e-6 (internal default used by ode45)
%        nref = (optional) ratio of the number of solution steps returned
%               by ode45 to those actually computed with the RK-45 method.
%               Default: nref = 1, meaning all returned values are from
%               steps of the RK-45 algorithm.  nref>1 causes ode45 to
%               interpolate nref additional "solution" values per step
%
% Output:    A table comparing the numerical and exact solutions

if nargin<1, rtol = 1e-3;  end
if nargin<2, atol = 1e-6;  end
if nargin<3, nref = 1;     end

%  Set tolerance and output refinement options.
options = odeset('RelTol',rtol,'AbsTol',atol,'Refine',nref);
tn = 1;  y0 = 1;
flops(0);  [t,y] = ode45('rhs2',tn,y0,options);  f = flops;
yex = y0*exp(-t);   emax = norm(y-yex,inf);

fprintf('      t      y_ode45    y_exact     error\n');
for k=1:length(t)
  fprintf('  %7.4f  %8.6f  %8.6f  %9.2e\n',t(k),y(k),yex(k),y(k)-yex(k));
end
fprintf('\nMax error =%12.4e rtol = %9.2e atol = %9.2e\n',emax,rtol,atol);
fprintf('%g flops\n',f);
plot(t,yex,'-',t,y,'o');
title(sprintf('Solution to dy/dt = -y;  Refine = %d',nref));
xlabel('t');  ylabel('y');
```

705

运行带缺省容差的demoODE45opts，可得到如下结果：

```
>> demoODE45opts
   x       y_ode45    y_exact    error
0.0000   1.000000   1.000000   0.00e+00
0.1000   0.904837   0.904837   2.97e-10
0.2000   0.818731   0.818731   5.38e-10
  ..
0.8000   0.449329   0.449329   1.18e-09
0.9000   0.406570   0.406570   1.20e-09
1.0000   0.367879   0.367879   1.21e-09

Max error = 1.21e-09 for rtol = 1.00e-03 atol = 1.00e-06
3450 flops
```

最大误差1.2×10^{-9}比由固定步长odeRK4程序（见例12.6）得到的最大误差还小两个数量级。这里，精度的增加不是由计算过程中h的改变产生的，ode45的每步都使用$h = 0.1$。因为ode45通过比较Runge-Kutta法的四阶和五阶解来监测LDE，在其每一步应该有五阶的解。可以比较四阶和五阶解之间的差异来修正四阶的解，以使结果的GDE小于$O(h^4)$。

注意，要得到相同的精度，在缺省容差和Refine=1下，ode45计算需要的浮点操作次数比odeRK4多3倍（再参考例12.6）。但是ode45的每一步（$h = 0.025$）却比odeRK4的（$h =$

0.1）小。对这个简单的ODE，固定步长算法具有优势，但是对更复杂的ODE，情况就不是这样了。

通过ode45传递附加参数 大多数ODE包含会影响解的参数。例如，描述指数式衰减

$$\frac{dy}{dt}=-\alpha y \tag{12-28}$$

的ODE（对比方程（12-26））就有参数α，它是时间常数的倒数。下面是计算此ODE右端表达式的一个简单函数rhsDecayNot，但是它并不正确。

```
function dydt = rhsDecayNot(t,y)
% rhsDecayNot Incorrect attempt to define rhs of dydt = -alpha*y
dydt = -alpha*y;
```

706

此m文件的问题当然就是α未定义。

内置的ODE函数能够将参数传给ODE定义函数。调用ode45的一般语法是：

```
[x, y] = ode45 (diffeq, tn, y0, options, args...)
```

其中*args...*是由逗号分开的参数列表，这些附加*args*只是通过ode45传给*diffeq*代表的m文件。如果可选参数要通过ode45传递，就需要*options*参数（见上节）。如果没有*options*要设置，*options*参数就使用空矩阵[]。在ODE定义程序中，必须提供附加的输入参数来接受传递参数，此程序的一般语法为：

```
function dydt = rhsFunction (t, y, flag, args...)
```

其中*args*是由ode45传递过来的输入参数的列表。当附加参数经过ode45传递时就需要参数*flag*，此参数还支持ode45更复杂的功能，这里不作介绍[⊖]。

使用程序清单12-7中的（正确的）rhsDecay函数，下面的语句可以调用ode45来解方程（12-28），其中初始条件为$y_0=10$。

```
>> tn = 4; y0 = 1; alpha = 5;
>> [t,y] = ode45('rhsDecay',tn,y0,[],alpha);
```

这种方法的优点是可以使用不同的alpha求解ODE，这样做只需要改变传给ode45的alpha值即可。

例12.9描述如何利用ode45函数的几个特征来求解一个更复杂、更有趣的一阶ODE问题。

例12.9 金属棒的热处理

本例示范了如何自动改变ode45的步长，以便更易于求解复杂的ODE，也展示了通过插值来改变方程解的精度的效果。在介绍数值解之前，需要给出这个物理问题的背景信息。

金属材料的强度由其制造过程中的化学成分和机械成形过程控制。融化的金属呈固态，在还没有完全冷却时，可以将其加工成条状、片状或杆状，然后将金属冷却，最后就要对其进行所谓的热处理，热处理过程就是要小心控制材料冷却时温度的变化。热处理的简单模型是固体金属的温度作为时间的函数的非线性一阶ODE[⊜]。

707

⊖ 在MATLAB提示符下键入help odefile可得到更多的信息。

⊜ 本例基于文献[39]第5章的章后习题。

程序清单12-7 函数demoODE45args使用ode45或带可选传递参数的积分法解一阶ODE。函数rhsDecay定义了ODE组的右端表达式，该表达式依赖于附加参数alpha的值

```
function demoODE45args(alpha)
% demoODE45args  Integrate dy/dt = -alpha*y;  y(0) = 1 with variable alpha
%
% Synopsis:  demoODE45
%            demoODE45(alpha)
%
% Input      alpha = (optional) decay rate coefficient.  Default:  alpha = 5
%                   Default: rtol = 1e-3 (internal default used by ode45)
%
% Output:    A table and plot comparing the numerical and exact solutions

if nargin<1, alpha = 2;  end

tn = 1;  y0 = 1;
[t,y] = ode45('rhsDecay',tn,y0,[],alpha);
yex = y0*exp(-alpha*t);
fprintf('      t       y_ode45    y_exact     error\n');
for k=1:length(t)
  fprintf('  %7.4f  %8.6f  %8.6f  %9.2e\n',t(k),y(k),yex(k),y(k)-yex(k));
end
fprintf('\nMax error = %10.2e for alpha = %9.2e\n',norm(y-yex,inf),alpha);
plot(t,yex,'-',t,y,'o');
title(sprintf('Solution to dy/dt = -%g*y',alpha));  xlabel('t');  ylabel('y');

function dydt = rhsDecay(t,y,flag,alpha)
% rhsDecay  Evaluate rhs of dy/dt = -alpha*y with a variable alpha.
%           "flag" parameter is required for compatability with ode45
dydt = -alpha*y;
```

708

图12-9 钢条的热处理

图12-9是一种特殊类型热处理的简图。其中，钢材已经加工成长条，将其加热可以释放加工过程中产生的应力。通过加热或其他方法去掉机械应力的过程称为退火。在这个特殊的退火过程中，钢条由通过它们的电流来加热，达到预期温度时，就关掉电流，并用风扇对钢条吹凉风。

在加热和冷却过程中，钢条通过对流和辐射交换热量。对流热传递是由于钢条周围的空气运动，这个过程可表示为：

$$Q_c = HA_s(T - T_a)$$

其中H是称为传热系数的经验常数，A_s是钢条的表面积，T是钢条的温度（假设内部是均衡的），T_a是空气温度。

辐射热传递是由于红外波段电磁辐射的作用，钢条与通过红外线能"看见"的任何表面交换热量。在本例中，我们假设钢条四周的所有表面温度都为T_a，即环境温度为T_a。那么辐射冷却的速率为：

$$Q_r = \varepsilon\sigma A_s(T^4 - T_a^4)$$

其中ε是钢条表面的发射率（$0 \leqslant \varepsilon \leqslant 1$），$\sigma$是Stefan-Boltzmann常数，其值为$5.67 \times 10^{-8}$ W/m^2K^4。当涉及辐射热传递时，所有的计算必须使用热力学温度（即开尔文（K）或Rankine温度（°R））。在针对此问题开发的m文件中，计算时要使用热力学温度为单位，输出时就转化成摄氏度，T（K）$= T$（°C）$+ 273.15$。

在任何时刻，由热量平衡得

$$mc\frac{\mathrm{d}T}{\mathrm{d}t} = Q_e - Q_c - Q_r$$

其中t是时间，m是钢条的质量，c是钢条比热容，Q_e是产生电热的速率，Q_c是对流冷却的速率，Q_r是辐射冷却的速率。将前面的关系式代入到热量平衡方程中，整理得

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{mc}\left\{Q_e - A_s\left[H(T - T_a) + \varepsilon\sigma(T^4 - T_a^4)\right]\right\} \tag{12-29}$$

709

为使此问题更有趣和现实，假设热处理过程分两个阶段。第一个阶段，加上电流，关掉提供凉风的风扇。第二个阶段，关闭电流，打开风扇。当凉风关闭时，对流冷却速度就大大地减小了。没有风扇，空气的运动就只能靠钢条附近空气增加的浮力（密度减小）来推动，这就叫自由对流（*free convection*），对应于打开风扇时的强制对流（forced convection）。这种几何形状下自由对流到强制对流转换的合理模型为：

$$H = \begin{cases} 15W/(m^2{}^\circ C) & t < t_i \quad (自由对流) \\ 100W/(m^2{}^\circ C) & t > t_i \quad (强制对流) \end{cases} \tag{12-30}$$

其中t_i是电流停止、风扇开启的时刻。

热处理模型由程序清单12-8中的demoSteel函数和程序清单12-9中的rhsSteelHeat函数实现。主程序demoSteel产生初始条件和控制参数，调用ode45来求解，并画出结果图，RhsSteelHeat计算方程（12-29）的右端表达式。检查demoSteel中对ode45的调用以及rhsSteelHeat函数的输入参数列表，可知问题中有很多参数要通过ode45传递。方程（12-30）中的两个传热系数存储在rhsSteelHeat的输入向量中，用来模拟从自由对流到强制对流的过渡。

在缺省参数下运行模拟程序，得到如下结果及图12-10中的左图。

```
>> demoSteel
rtol = 1.00e-03 atol = 1.00e-06 Tmax = 656.86
```

不对输出进行插值（即令nref=1）运行该模拟程序，得到如下结果及图12-10中的右图。

```
>> demoSteel(1)
rtol = 1.00e-03 atol = 1.00e-06 Tmax = 656.86
```

两个结果是相同的，只是ode45输出点的数目不同。

程序清单12-8 函数demoSteel模拟钢条的热处理过程（另见程序清单12-9中的rhsSteelHeat）

```
function [tout,Tout] = demoSteel(nref,rtol,atol)
% demoSteel  Solve ODE describing heat treating of a steel bar using ode45
%
% Synopsis:  demoSteel                     [t,T] = demoSteel
%            demoSteel(nref)               [t,T] = demoSteel(nref)
%            demoSteel(nref,rtol)          [t,T] = demoSteel(nref,rtol)
%            demoSteel(nref,rtol,atol)     [t,T] = demoSteel(nref,rtol,atol)
%
% Input:     nref = (optional) number of interpolations between steps.
%                    Default:  nref = 4 (internal default used by ode45)
%            rtol = (optional) relative tolerance used by ode45
%                   Default: tol = 1e-3 (internal default used by ode45)
%            atol = (optional) absolute tolerance used by ode45
%                   Default: tol = 1e-6 (internal default used by ode45)
%
% Output:    Plot of temperature of the bar versus time.
%            t = (optional) vector of times at which solution is obtained
%            T = (optional) vector of temperatures from numerical solution

if nargin<1,  nref = 4;     end    % Process optional input arguments
if nargin<2,  rtol = 1e-3;  end
if nargin<3,  atol = 1e-6;  end

len = 1;     dia = 1e-2;         % bar length and diameter, meters
rho = 7822;  c = 444;            % density (kg/m^3) and heat capacity (J/kg/K)
As = pi*dia*len;                 % surface area of bar, m^2, neglect ends
mc = len*0.25*pi*dia^2*rho*c;    % mc = rho*volume*c
emiss = 0.7;                     % emissivity of bar
htc = [15; 100];                 % heat transfer coefficients, W/m^2/C
Ta = 21 + 273.15;                % ambient temperature, K
tcool = 70;                      % begin cooling at tcool seconds
tf = 3*tcool;                    % total simulation time, seconds
QV - 3000;                       % rate of electrical heat generation, W

% --- Set tolerance and refinement options and solve the ODE
options = odeset('RelTol',rtol,'AbsTol',atol,'Refine',nref);
[t,T] = ode45('rhsSteelHeat',tf,Ta,options,mc,QV,tcool,htc,As,Ta,emiss);

T = T - 273.15;                 % convert kelvins to Celsius
f = figure;  plot(t,T,'+');     % open new figure window and plot
xlabel('Time  (s)');  ylabel('Temperature (C)');
title(sprintf('nref = %d,  rtol = %9.1e,  atol = %9.1e',nref,rtol,atol));
fprintf('rtol = %9.2e  atol = %9.2e  Tmax = %7.2f\n',rtol,atol,max(T));

if nargout>1,   tout = t;  Tout = T;  end   % Optional return variables
```

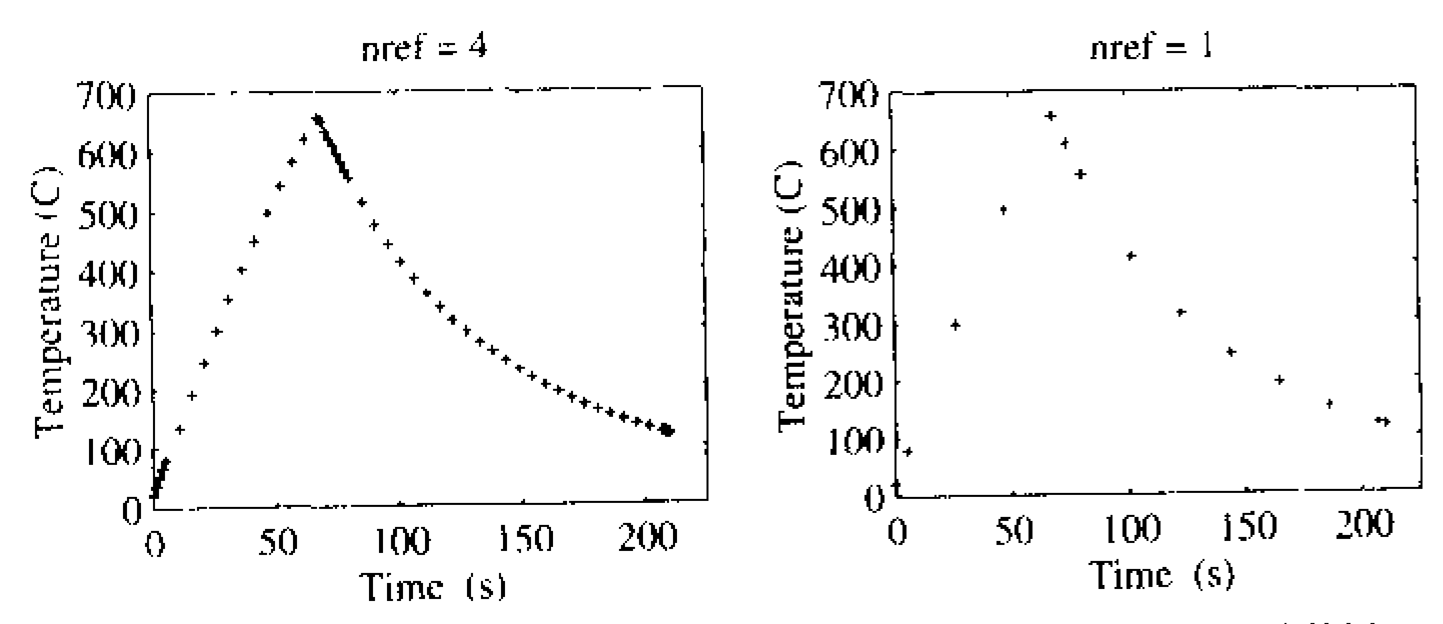
710
711

程序清单12-9 函数rhsSteelHeat定义ODE的右端表达式以模拟钢条的热处理

```
function dTdt = rhsSteelHeat(t,T,flag,mc,QV,tcool,htc,As,Ta,emiss)
% rhsSteelHeat  Right hand side of first order ODE for heat treating simulation
%
% Synopsis  dTdt = rhsSteelHeat(t,T,flag,mc,QV,tcool,htc,As,Ta,emiss)
```

```
%
% Input:  t     = time (sec)
%         T     = current estimate of bar temperature (K)
%         flag  = (not used) placeholder for compatibility with ode45
%         mc    = product of mass and specific heat capacity (J/K)
%         QV    = rate volumetric heat addition (W)
%         tcool = time (sec) at which heating is stopped and cooling begins
%         htc   = vector of convective heat transfer coefficients (W/m^2/C)
%         As    = surface area of the bar (m^2)
%         Ta    = ambient tempeature (K)
%         emiss = emissivity of the bar (dimensionless)
%
% Output:  dTdt = rate of increase of temperature with time.

if t<tcool
  QVol = QV;  hh = htc(1);     %  heating phase, natural convection
else
  QVol = 0;   hh = htc(2);     %  cooling phase, forced convection
end
dTdt = (1/mc)*( QVol - As*( hh*(T - Ta) + emiss*(5.67e-8)*(T^4 - Ta^4) ) );
```



712

图12-10 热处理模拟结果。左图描述了使用缺省"options"参数下ode45得到的解。右图是nref = 1（即无输出插值）时ode45得到的解。两图中求数值解的点相同

12.5 联立ODE组

迄今为止，所介绍的算法只适用于单个、一阶的ODE。对其稍微改进，同样的算法就可以适用于一阶联立ODE组。考虑下面的常微分方程对:

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2) \tag{12-31}$$

$$\frac{dy_2}{dt} = f_2(t, y_1, y_2) \tag{12-32}$$

这些方程是联立的，因为明显地y_1的方程依赖于y_2，反过来也一样。

本节主要介绍一阶联立ODE组，并用四阶Runge-Kutta法来展示基本的求解方法。高阶ODE与一阶ODE组等价，本节中也介绍了求解高阶ODE的方法。

12.5.1 联立ODE组的RK-4算法

四阶Runge-Kutta法需要在每一步求四个斜率估计值（dy/dt的值）。对方程（12-31）来说，任何t上的dy_i/dx值都依赖于y_1和y_2，因此，需要同步地计算两个方程的中间斜率，这很重要。换言之，基本RK-4法在每个方程中都要用k_1，k_2，k_3和k_4，但是每个k_i（i = 1，…，4）都依赖于t以及潜在地依赖于所有的y值。要合适地应用此算法，所有微分方程中的k_1必须在k_2之前计算出来。

考虑两个联立方程的积分，其中（$j-1$）步的值已知。令$k_{1,i}$为第i个方程第1个Runge-Kutta系数（斜率），$k_{2,i}$为第i个方程的第2个Runge-Kutta系数，依次类推。要进展到第j步，首先要计算

$$k_{1,1} = f_1(t_j, y_{j,1}, y_{j,2})$$
$$k_{1,2} = f_2(t_j, y_{j,1}, y_{j,2})$$

然后计算

$$k_{2,1} = f_1\left(\left(t_j + \frac{h}{2}\right), \left(y_{j,1} + \frac{h}{2}k_{1,1}\right), (y_{j,2} + \frac{h}{2}k_{1,2})\right)$$
$$k_{2,2} = f_2\left(\left(t_j + \frac{h}{2}\right), \left(y_{j,1} + \frac{h}{2}k_{1,1}\right), (y_{j,2} + \frac{h}{2}k_{1,2})\right)$$

依次类推

[713]

$$k_{3,1} = f_1\left(\left(t_j + \frac{h}{2}\right), \left(y_{j,1} + \frac{h}{2}k_{2,1}\right), (y_{j,2} + \frac{h}{2}k_{2,2})\right)$$
$$k_{3,2} = f_2\left(\left(t_j + \frac{h}{2}\right), \left(y_{j,1} + \frac{h}{2}k_{2,1}\right), (y_{j,2} + \frac{h}{2}k_{2,2})\right)$$
$$k_{4,1} = f_1((t_j + h), (y_{j,1} + hk_{3,1}), (y_{j,2} + hk_{3,2}))$$
$$k_{4,2} = f_2((t_j + h), (y_{j,1} + hk_{3,1}), (y_{j,2} + hk_{3,2}))$$

只有所有的中间斜率值都计算出来之后，因变量才进行到下一步：

$$y_{j,1} = y_{j-1,1} + h\left(\frac{k_{1,1}}{6} + \frac{k_{2,1}}{3} + \frac{k_{3,1}}{3} + \frac{k_{4,1}}{6}\right)$$
$$y_{j,2} = y_{j-1,2} + h\left(\frac{k_{1,2}}{6} + \frac{k_{2,2}}{3} + \frac{k_{3,2}}{3} + \frac{k_{4,2}}{6}\right)$$

这些计算可以使用循环结构来实现，如程序清单12-10中的odeRK4sys函数所示。写出odeRK4sys函数是为了清楚表示计算思想，它并非是最佳的MATLAB代码实例。函数odeRK4sysv（此处并未列出，包含在NMM工具箱中）将所有的for k = 1:neq循环写成向量表达式，提高了效率。注意odeRK4sysv也支持varargin工具，因此不需要使用全局变量将参数传给ODE定义程序。

函数odeRK4sys和odeRK4sysv是通用的，二者足够处理任何数目的微分方程组。方程组中方程的数目neq由向量y0的长度决定，此向量中包含方程组的初始条件。为解方程组，用户所提供的ODE右端表达式的定义程序必须要能处理向量，其输入是标量t和y值向量，输出是dy/dt值向量，向量y和dy/dt的长度必须等于neq。

内置的ode45函数使用与odeRK4sys一样的方法处理方程组。实际上，建立

odeRK4sys 一个重要的原因是要说明在对方程组积分时如何用ode45来写ODE定义程序。ODE定义程序的细节在例12.10和例12.11中加以示范。

例12.10 联立方程组

在本例中，将对如下方程组求数值解：

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = -y_1 e^{(1-t)} + 0.8y_2, \qquad y_1(0) = 0$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = y_1 - y_2^3, \qquad y_2(0) = 2$$

714

程序清单12-10 函数odeRK4sys非向量化地实现了四阶Runge-Kutta法，此函数的向量化版本是odeRK4sysv函数，它在NMM工具箱的ode目录下

```
function [t,y] = odeRK4sys(diffeq,tn,h,y0)
% odeRK4sys  Fourth order Runge-Kutta method for systems of first order ODEs
%            Nonvectorized version
%
% Synopsis:  [t,y] = odeRK4sys(diffeq,tn,h,y0)
%
% Input:     diffeq = (string) name of the m-file that evaluates the right
%                     hand side of the ODE system written in standard form.
%            tn     = stoping value of the independent variable
%            h      = stepsize for advancing the independent variable
%            y0     = vector of the dependent variable values at t = 0
%
% Output:    t = vector of independent variable values:  t(j) = (j-1)*h
%            y = matrix of dependent variables values, one column for each
%                state variable.  Each row is from a different time step.

t = (0:h:tn)';           %  Column vector of elements with spacing h
nt = length(t);          %  number of steps
neq = length(y0);        %  number of equations that are simultaneously advanced
y = zeros(nt,neq);       %  Preallocate y for speed
y(1,:) = y0(:)';         %  Assign IC. y0(:) is column, y0(:)' is row vector

h2 = h/2;  h3 = h/3;  h6 = h/6;   %  Avoid repeated evaluation of constants
k1 = zeros(neq,1);  k2 = k1;      %  Preallocate memory for the Runge-Kutta
k3 = k1;  k4 = k1;  ytemp = k1;   %  coefficients and a temporary vector

%  Outer loop for all steps.  j = time step index;  n = equation number index
for j=2:nt
   told = t(j-1);   yold = y(j-1,:)';   %  Temp variables, yold is column vector
   k1 = feval(diffeq,told,yold);        %  Slopes at the starting point
   for n=1:neq
     ytemp(n) = yold(n) + h2*k1(n);     %  Estimate all y's at midpoint
   end
   k2 = feval(diffeq,told+h2,ytemp);    %  1st estimate of slopes at midpoint
   for n=1:neq
     ytemp(n) = yold(n) + h2*k2(n);     %  2nd estimate of all y's at midpoint
   end
   k3 = feval(diffeq,told+h2,ytemp);    %  2nd estimate of slopes at midpoint
   for n=1:neq
     ytemp(n) = yold(n) + h*k3(n);      %  Estimate y at end point
```

```
    end
    k4 = feval(diffeq,told+h,ytemp);      % Estimate of slopes at endpoint
    for n=1:neq                           % Simultaneously advance all y's
      y(j,n) = yold(n) + h6*(k1(n)+k4(n)) + h3*(k2(n)+k3(n));
    end
  end
```

715

程序清单12-11　函数demoSystem使用ode45或odeRK4sysv对一阶联立ODE组积分。函数rhsSys定义了ODE组的右端表达式

```
function demoSystem
% demoSystem  Solve system of two coupled first order ODEs
%
% Synopsis:  demoSystem
%
% Input:     none
%
% Output:    Plot of solution

y0 = [0; 2];     % Initial conditions stored in a column vector
tn = 3;
%  [t,y] = ode45('rhsSys',tn,y0);               %  ode45 solution
[t,y] = odeRK4sysv('rhsSys',tn,0.1,y0);  %  odeRK4sys solution
plot(t,y(:,1),'+',t,y(:,2),'o'),
xlabel('t');  ylabel('y_1  and  y_2');  legend('y_1','y_2');

function dydt = rhsSys(t,y)
% rhsSys  Right hand side vector for two, coupled, first order ODEs
dydt = [ -y(1)*exp(1-t) + 0.8*y(2);
          y(1) - y(2)^3];                 %  a column vector
```

程序清单12-11下半部分的rhsSys函数计算出方程组的右端表达式。注意，返回变量dydt是ode45需要的列向量。此方程的解及其图示可由以下语句得到：

```
>> y0 = [0; 2];                      % ICs are stored in a *column* vector
>> [t,y] = ode45('rhsSys',3,y0);     % Solution on interval 0 <= t <= 3
>> plot(t,y(:,1),'+',t,y(:,2),'o');
```

结果如图12-11所示。

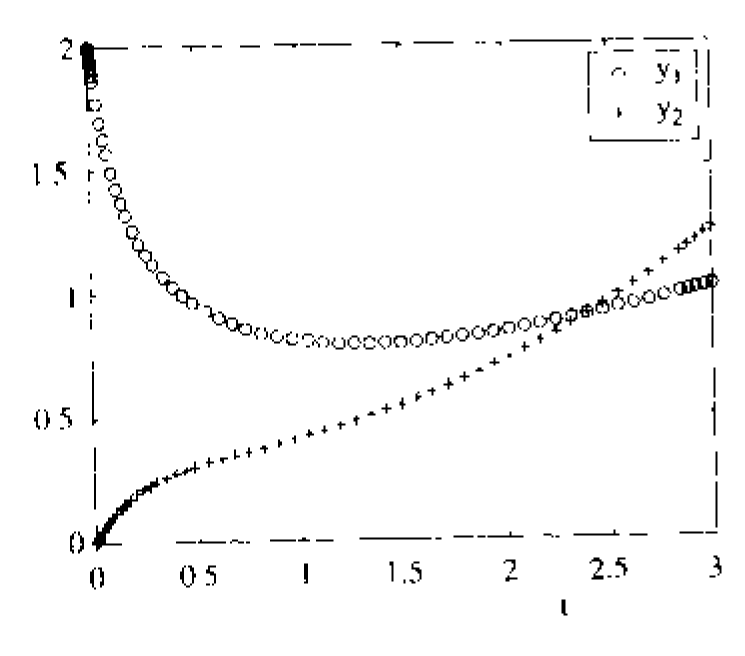


图12-11　联立一阶ODE组$dy_1/dt = -y_1\exp(1-t) + 0.8y_2$，$dy_2/dt = y_1 - y_2^3$的结果图

例12.11　猎食者-猎物方程

本例示范了如何解生物系统模型的一阶联立ODE组。在这之前，先给出一些背景信息，数值解本身其实并不难。

考虑两个互相依赖的动物种群的动态数量，其中，猎物是另一种族——猎食者的食物来源，下面就对此类问题建模。例如，在一个孤立的小岛上的兔子（猎物）和狼（猎食者）的种群。令猎物的种群数量为$p_1(t)$，猎食者的种群数量为$p_2(t)$，则两个种群的增长速率模型为 716

$$\frac{dp_1}{dt}=\alpha_1 p_1-\delta_1 p_1 p_2 \tag{12-33}$$

$$\frac{dp_2}{dt}=\alpha_2 p_1 p_2-\delta_2 p_2 \tag{12-34}$$

其中α_1和α_2是增长率系数，δ_1和δ_2是两个种群的死亡率系数。

方程（12-33）的右端第一项表示猎物的繁衍，假设这只跟配偶的数量有关，并假设猎物不缺乏食物；右端第二项表示猎物的死亡率，它随着猎食者（捕食猎物）的数量和猎物种群数量的增大而增大，系数δ_1反映了猎食者的捕食效率。

方程（12-34）是有关猎食者数量增长的微分方程。猎食者的繁衍与其数量有关，但也受食物（p_1）的限制，系数α_2就描述了猎食者的繁衍以及猎物的营养价值；猎食者的死亡率与其自身的种群数量成直接比例，与猎物的种群数量无关。当猎食者种群增长时，它们就会更快地消耗掉其食物供应，作为主要或次要的死亡原因，饥饿的重要性也就凸现出来。

解此微分方程组需要初始条件$p_1(0)$和$p_2(0)$、繁衍和死亡率系数α_1、α_2、δ_1和δ_2，以及一种对微分方程组积分的方法。初始条件和繁衍、死亡率系数来自对被研究物种的了解，数值积分法必须能够对微分方程组进行积分。

猎食者-猎物模型的特定问题代码包含在两个m文件中。程序清单12-12中的主程序demoPredPrey定义了初始条件，调用ode45并画图。给出标量t中存储的时间、存储在向量p中的种群数量p_1和p_2，以及参数向量alpha和delta，就可以用程序清单12-12下半部分的ODE定义程序rhspop2计算出方程（12-33）和方程（12-34）的右端表达式。dp/dt的值返回到ode45的dpdt向量中。时间不作为方程（12-33）和方程（12-34）的明确参数出现，但是它必须是rhspop2的输入参数，才能与ode45兼容。此外，为和ode45兼容，还需要flag参数，但它不起作用。 717

试着运行猎食者-猎物模型程序，可得到如图12-12所示的结果。这些结果在缺省值

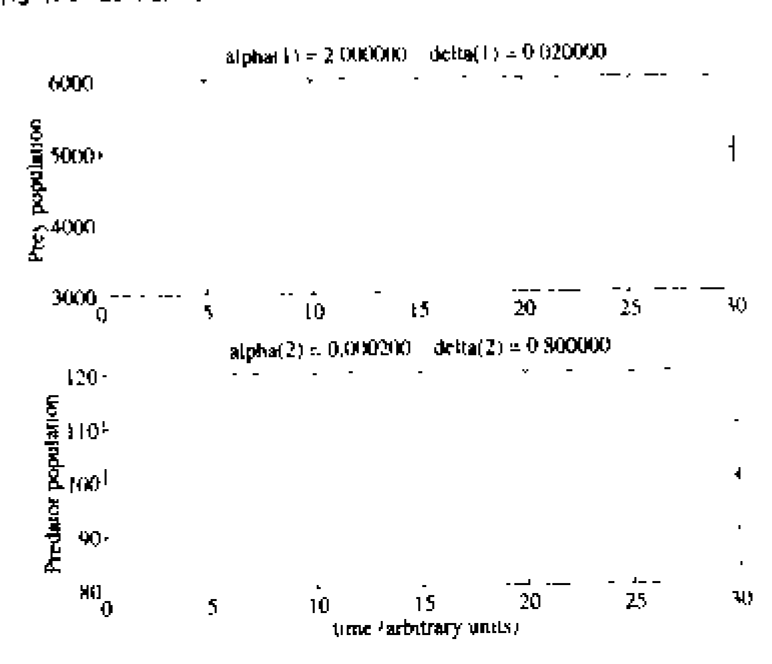


图12-12　在$\alpha_1=2$、$\delta_1=0.02$、$\alpha_2=0.0002$和$\delta_2=0.8$时，初始种群数量为5000猎物和100猎食者的猎食者-猎物模型的计算结果

p1start、p2start和tstop下得到，键入

```
>> demoPredPrey
```

在demoPredPrey中，将odeRK4sysv前面的注释符%去掉，并将ode45的调用语句变成注释，就可以用odeRK4sysv积分程序代替ode45。注意使用odeRK4sysv积分的结果受h影响很大。一般地，最好不要用odeRK4sysv，建议使用ode45，因为它使用自适应步长算法，可以在每一步中保持高精度。

718

程序清单12-12　函数demoPredPrey使用ode45或odeRK4sysv来对两物种的猎食者-猎物模型积分。函数rhsPop2定义了ODE组的右端表达式

```
function demoPredprey(p1start,p2start,tstop)
% demoPredprey  Coupled ODEs for a two-species predator-prey simulation
%
% Synopsis:  demoPredprey
%            demoPredprey(p1start,p2start,tstop)
%
% Input:     pstart1 = (optional) initial population of species 1
%            pstart2 = (optional) initial population of species 2
%            tstop   = (optional) duration of the simulation
%
% Output:    Plot of populations versus time

if nargin<1,  p1start = 5000;  end
if nargin<2,  p2start = 100;   end
if nargin<3,  tstop = 30;      end

% ---  Parameters to be passed to rhspop2
alpha = [ 2.0  0.0002 ];      delta = [ 0.02  0.8 ];

p0 = [p1start; p2start];                                % Initial conditions
[t,p] = ode45('rhspop2',tstop,p0,[],alpha,delta);  % Solution with ode45
% [t,p] = rk4sysv('rhspop2',tstop,tstop/500,p0,alpha,delta);  % with rk4sysv

% --- Plot the results
subplot(2,1,1);  plot(t,p(:,1));   grid;  ylabel('Prey population');
title(sprintf('alpha(1) = %f    delta(1) = %f',alpha(1),delta(1)));

subplot(2,1,2);  plot(t,p(:,2));   grid;
xlabel('time (arbitrary units)');  ylabel('Predator population');
title(sprintf('alpha(2) = %f    delta(2) = %f',alpha(2),delta(2)));

function dpdt = rhsPop2(t,p,flag,alpha,delta)
% rhsPop2  Right hand sides of coupled ODEs for 2 species predator-prey system
%
% Synopis:  dpdt = rhsPop2(t,p,flag,alpha,delta)
%
% Input:    t = time. Not used in this m-file, but needed by ode45
%           p = vector (length 2) of populations of species 1 and 2
%           flag  = (not used) placeholder for compatibility with ode45
%           alpha = vector (length 2) of growth coefficients
%           delta = vector (length 2) of mortality coefficients
%
```

```
% Output    dpdt = vector of dp/dt values
dpdt = [ alpha(1)*p(1)        - delta(1)*p(1)*p(2);
         alpha(2)*p(1)*p(2) - delta(2)*p(2);      ];
```

719

12.5.2 高阶微分方程

对高阶微分方程进行数值积分，必须将方程进行数学变换，变成等价的一阶ODE组。考虑二阶ODE

$$\frac{d^2u}{dt^2}+\alpha\frac{du}{dt}+\beta u=f(t) \tag{12-35}$$

它决定了函数$u=u(t)$。引进两个新的状态变量y_1和y_2，令

$$y_1\equiv u \qquad y_2\equiv\frac{du}{dt}$$

分别取y_1和y_2对t的导数，得

$$\frac{dy_1}{dt}=\frac{du}{dt}=y_2$$

$$\frac{dy_2}{dt}=\frac{d^2u}{dt^2}=f(t)-\alpha\frac{du}{dt}-\beta u=f(t)-\alpha y_2-\beta y_1$$

因此，含两个方程的一阶联立ODE

$$\frac{dy_1}{dt}=y_2 \tag{12-36}$$

$$\frac{dy_2}{dt}=f(t)-\alpha y_2-\beta y_1 \tag{12-37}$$

与方程（12-35）中单个的二阶ODE等价。

任何n阶ODE可以转化成数学上等价的含n个方程的一阶联立ODE组。已知

$$\frac{d^nu}{dx^n}=f(t,u)$$

其变换后为

定义y_i	关于y_i的ODE
$y_1\equiv u$	$\frac{dy_1}{dt}=y_2$
$y_2\equiv\frac{du}{dt}$	$\frac{dy_2}{dt}=y_3$
$y_3\equiv\frac{d^2u}{dt^2}$	$\frac{dy_3}{dt}=y_4$
⋮	⋮
$y_n\equiv\frac{d^{n-1}u}{dt^{n-1}}$	$\frac{dy_n}{dt}=f(t,u)$

720

通常将y_i称为方程组的状态变量。当单个高阶ODE转化成一阶方程组后，微分方程中除了最后一个外其他的右端表达式中都只包含另一个状态变量。

例12.12　二阶机械方程

本例展示了如何将一个二阶的ODE转化成含有两个方程的一阶ODE组。此数学方程描述了一个普通的机械系统模型。这里首先推导机械系统方程，然后再用ode45得出数值解。

考虑图12-13所示的弹簧减震器。一个随时间变化的外力施加到质量为m的物体上使其挤压弹簧，积累能量，弹簧施加一个恢复力，释放能量。物体的受力平衡方程为

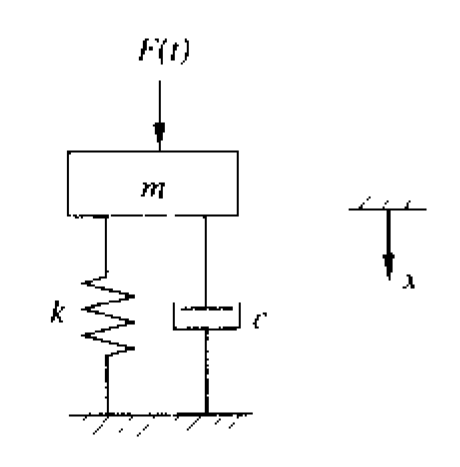


图12-13　一个普通的弹簧减震器系统

$$\sum F = ma$$

其中$\sum F$是施加到物体上的合力，a是物体的加速度。弹簧施加的恢复力施加在x轴的反方向，与弹簧压缩的距离成比例。减震器挡在物体的运动方向上，减震力随物体速度的增加而增加。假设这是一个标准的减震模型，外力与物体速度成正比。弹簧力和减震力分别为

$$F_{\text{spring}} = -kx, \qquad F_{\text{damper}} = -c\dot{x}$$

其中k是弹性常数，$\dot{x} \equiv \mathrm{d}x/\mathrm{d}t$是物体的瞬时速度。将前面的方程代入受力平衡方程，得

$$F(t) - kx - c\dot{x} = m\ddot{x}$$

其中$\ddot{x} = \mathrm{d}^2x/\mathrm{d}t^2$是物体的加速度。以上方程通常写成如下形式

$$\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = \frac{F}{m} \qquad (12\text{-}38)$$

[721] 其中ζ是无量纲的减震系数，ω_n是固有频率：

$$\zeta \equiv \frac{c}{2\sqrt{km}}, \qquad \omega_n \equiv \sqrt{k/m}$$

使用如下变换

$$y_1 \equiv x, \qquad y_2 \equiv \dot{x}$$

方程（12-38）可以写成等价的一阶ODE组，得

$$\frac{\mathrm{d}y_1}{\mathrm{d}t} = y_2 \qquad (12\text{-}39)$$

$$\frac{\mathrm{d}y_2}{\mathrm{d}t} = \frac{F}{m} - 2\zeta\omega_n y_2 - \omega_n^2 y_1 \qquad (12\text{-}40)$$

这两个方程组成了一个一阶联立ODE组。

受力函数有很多类型。为简便起见，使用一个阶梯函数来说明此二阶方程组的模型如何工作。阶梯函数相当于某一时刻在图12-13所示的物体上放置另一个质量为m_s的物体。这样，施加力的大小就是$m_s g$，其中g是重力加速度（在SI单位中为9.8m/s²）。在t_0时刻发生的阶梯函数可以描述为

$$F(t)=\begin{cases}0, & t<t_0\\ F_0, & t\geqslant t_0\end{cases}$$

其中F_0是阶梯的大小。

方程（12-39）和方程（12-40）中二阶方程组的数值模型由程序清单12-13中的demoSmd和rhssmd函数实现。主程序demoSmd设置初始条件，调用ode45并画出结果图；函数rhssmd定义需要积分的方程组。已知t，y_1和y_2的值，rhssmd可以计算出方程（12-39）和方程（12-40）右端表达式的值。参数zeta、omegan和a0=F_0/m通过ode45传给rhssmd，如12.4.1节所示。

在命令窗口键入

```
>> demoSmd
```

可得图12-14。系统对阶梯函数的瞬时响应依赖于系统参数ζ和ω_n的值。瞬时响应衰弱后，系统达到一种新的平衡状态，此状态与在弹簧上面加上质量为$m+m_s$的物体后得到的静态偏移状态相同。

722

程序清单12-13　函数demoSmd和rhssmd求解描述弹簧减震器系统的ODE

```
function demoSmd(zeta,omegan,tstop)
% demoSmd   Second order system of ODEs for a spring-mass-damper system
%
% Synopsis:  smdsys(zeta,omegan,tstop)
%
% Input:     zeta   = (optional) damping ratio;  Default:  zeta = 0.1
%            omegan = (optional) natural frequency;  Default: omegan = 35
%            tstop  = (optional) stopping time;  Default:  tstop = 1.5
%
% Output:    plot of displacement and velocity versus time

if nargin<1,  zeta = 0.1;    end
if nargin<2,  omegan = 35;   end
if nargin<3,  tstop = 1.5;   end

y0 = [0; 0];    a0 = 9.8;      %  Initial conditions and one g force/mass
[t,y] = ode45('rhssmd',tstop,y0,[],zeta,omegan,a0);

subplot(2,1,1);
plot(t,y(:,1));  ylabel('Displacement');  grid;
title(sprintf('zeta = %5.3f   omegan = %5.1f',zeta,omegan));
subplot(2,1,2);
plot(t,y(:,2));  xlabel('Time (s)');  ylabel('Velocity');  grid;

function dydt = rhsSmd(t,y,flag,zeta,omegan,a0)
% rhsSmd  Right-hand sides of coupled ODEs for a spring-mass-damper system
%
% Synopsis:  dydt = rhsSmd(t,x,flag,zeta,omegan,a0)
%
% Input:     t      = time, the independent variable
%            y      = vector (length 2) of dependent variables
%                     y(1) = displacement and y(2) = velocity
%            flag   = dummy argument for compatibility with ode45
%            zeta   = damping ratio (dimensionless)
```

```
%           omegan = natural frequency (rad/s)
%           a0     = input force per unit mass
%
% Output:   dydt = column vector of dy(i)/dt values

if t<=0, fonm = 0.0;
else,    fonm = a0;     %  Force/mass (acceleration)
end

dydt = [ y(2);  fonm - 2*zeta*omegan*y(2) - omegan*omegan*y(1)];
```

723

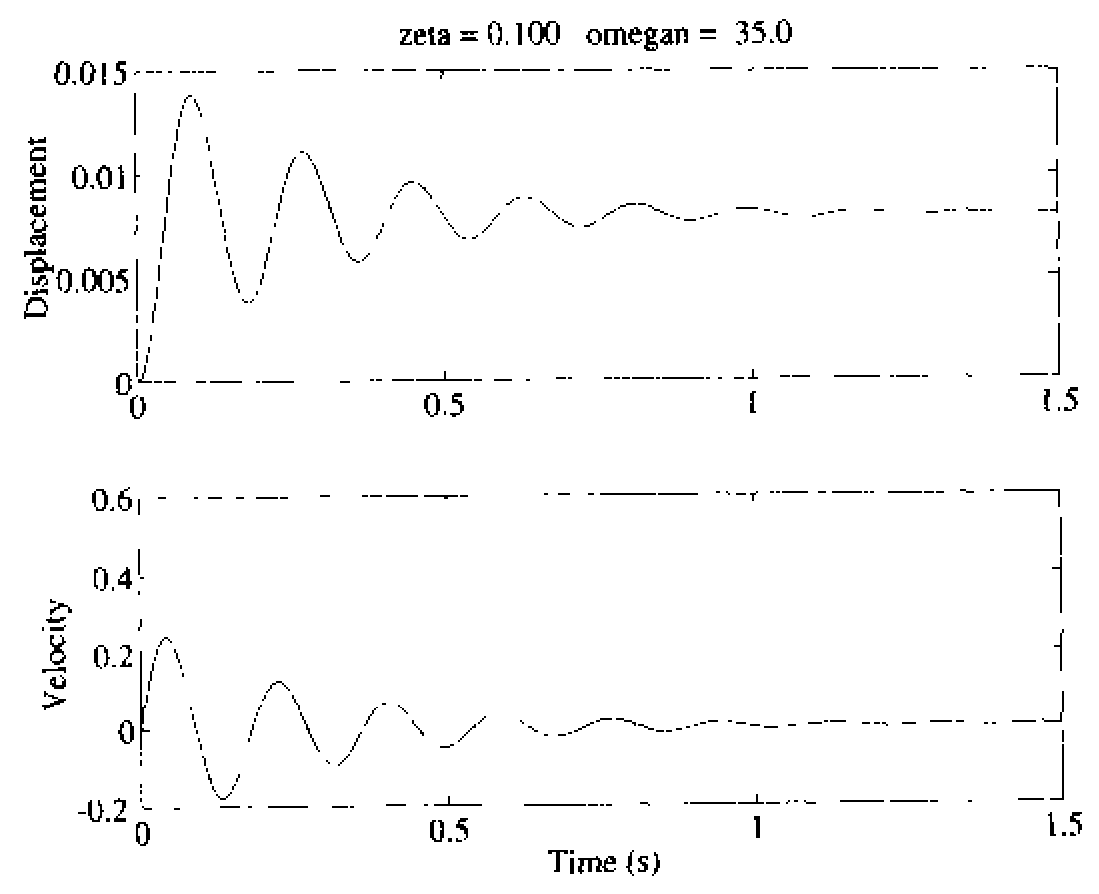


图12-14 $\zeta=0.1$、$\omega_n=35$（rad/s）时二阶机械系统的阶梯响应。初始条件为 $x=\dot{x}=0$

12.6 其他主题

本章只介绍了常微分方程的数值解法，另外还有一些重要的主题没有涉及到，例如ODE积分法的稳定性、多步法和刚性方程。一个算法的稳定性就是精确解不变时，其数值解在大小上增长的倾向（或没有这种倾向）。给定一个足够大的时间步，本章介绍的所有单步法的结果都会变得不稳定。

多步法是使用多个步长上的数值解来估计下一步的值。本章介绍的所有算法都是单步法，虽然其中大部分算法都在每一步上计算多个斜率值。内置的ode115就是一个多步法程序，它能自适应地选择前面一定步数的解来求当前步的解。

ODE的刚性是指变化的多重尺度。例如一些化学反应的数学模型就显示出随时间的改变反应速率或快或慢的现象。要追踪快速过程，就需要小的时间步，要解决慢速过程，就需要很大的总体积分时间。如果一个方程组的变化速率太过于不一致，以致于像RK-4这样的标准方法都不能满足时间量程长短变化的要求，就说这个方程组是刚性的。如果一个ODE积分算法需要的时间步太小，已经不适合数值求解（即解不切实际），那么这个算法的稳定性就会因刚性问题变得很重要。MATLAB工具箱中的ode23s和ode115s程序就是专门用来处理刚性问题的。

724

关于稳定性和多步法的更多信息，Shampine et al.[65]中有所介绍。Moler[56]对刚性和MATLAB中处理此问题的程序作了简略描述。很多介绍数值计算的书都对刚性作一些简单的讲解。Finlayson[23]对此提供了更完整的讨论，并使用了化学工程中出现的刚性方程组作为范例。

12.7 小结

本章介绍了常微分方程数值求解的多种算法，其中欧拉法是单步算法族中最简单但最不精确的方法。

由于欧拉法很简单，因此其局部和全局离散误差易于分析。同种类型的分析法也可用于其他单步法中。局部离散误差建立了方程（12-15）和方程（12-24）所定义方法的阶，但全局离散误差在实际问题中更有用，因为它提供了求解区间中的最大误差的估计。对单步法来说，局部离散误差的阶数与全局离散误差的阶数相同。

中点法和固定步长的RK-4法比欧拉法离散误差更小，其精度的增加是通过在每一步计算多个斜率得到的。增加这些额外计算工作是值得的，因为在保持相同精度的同时，可以采用更大的时间步，而得到更小的离散误差。本章中的数值实验证明：给定某一个要求的精度，中点法远比欧拉法更有效，而固定步长的RK-4法又远比中点法和欧拉法更有效。表12-2列出了一些m文件，示范了一些简单算法，并比较了它们之间的不同性能。

MATLAB工具箱包含了多个求解ODE的程序。在求解实际问题时，强烈推荐表12-3中列出的内置程序。其中ode23和ode45函数包含了对变步长Runge-Kutta法的复杂精致的实现，用户在使用这些函数时并不指定步长，而是给出每一步要满足的容差，缺省容差提供了合理的开始点。在遇到新的ODE问题要解决时，应该首先想到尝试ode45函数。

本章前半部分介绍了欧拉法、中点法、Heun法和固定步长的RK-4法，这些方法在开发MATLAB程序时并不提倡使用。这些简单的算法在证明算法思想时很有用，但是，正如本章的例子所示，若要以最少的计算工作得到最精确的解，这些简单的程序根本不能和内置的函数相比。这里介绍这些简单的算法，只是作为一种手段来推导表12-3所列出的复杂程序背后的逻辑和理论。

本章中介绍的所有的ODE积分程序都需要用户提供一个m文件来计算所求解微分方程的右端表达式。本章也描述了很多ODE的例子，以及计算这些ODE右端的m文件，如表12-4所示。

725

表12-2 本章开发的用来实现和比较ODE积分算法的函数。小节中带N.A.（不可用）的说明函数没有在本章中列出，但包含在NMM工具箱中

函 数	小 节	描 述
compEM	12.3.1	通过解$dy/dt=-y$；$y(0)=1$比较欧拉法和中点法的浮点操作次数和精度
compEMRK4	NA	通过解$dy/dt=-y$；$y(0)=1$比较欧拉法、中点法和RK-4法的浮点操作次数和精度
demoEuler	12.2.1	使用欧拉法对$dy/dt=t-2y$；$y(0)=1$积分
demoODE45	NA	使用ode45法来对$dy/dt=-y$；$y(0)=1$积分
demoODE45args	12.4.1	使用ode45法来对带变量α的方程$dy/dt=-\alpha y$；$y(0)=1$积分
demoODE45opts	12.4.1	使用ode45法和用户自定义收敛容差参数来对$dy/dt=-y$；$y(0)=1$积分

（续）

函　数	小　节	描　述
demoPredPrey	12.5.1	使用ode45法解两物种的猎食者-猎物模型的联立ODE组
demoSmd	12.5.2	使用ode45法解弹簧减震器系统的二阶ODE
demoSteel	12.5	解一个描述钢条热处理过程的一阶ODE
demoSystem	12.5.1	使用ode45法解一阶联立ODE组$dy_1/dx = -y_1\exp(1-x)+0.8y_2$; $dy_2/dx = y_1 - y_2^3$;$y_1(0)=1$; $y_2(0)=2$
odeEuler	12.2.1	使用欧拉法来对单个一阶ODE积分
odeMidpt	12.3.1	使用中点法对单个一阶ODE积分
odeRK4	12.3.3	使用四阶Runge-Kutta法对单个一阶ODE积分
odeRK4sys	12.5.1	使用固定步长的四阶Runge-Kutta法对单个一阶ODE积分（非向量化版）
odeRK4sysv	NA	使用固定步长的四阶Runge-Kutta法对单个一阶ODE积分（向量化版，提供传递参数）

表12-3　求解ODE的MATLAB内置程序。这些函数有相同的调用语法，但有不同的性能特点

函　数	描　述
ode113	对非刚性ODE组的可变阶求解方法，在1到13阶上使用显式的预测-更正法（predictor-corrector）
ode15s	对可变阶的刚性ODE组的多步解法，使用1到5阶的隐式多步法
ode23	非刚性ODE组的低阶自适应步长法程序，使用2阶和3阶的Runge-Kutta法
ode23s	中度刚性ODE组的低阶自适应步长法程序，使用2阶和3阶的Runge-Kutta法
ode45	非刚性ODE组的高阶自适应步长程序，使用4阶和5阶的Runge-Kutta法

726

表12-4　本章开发的用于计算ODE右端表达式的函数

函　数	小　节	描　述
rhs1	12.2.1	计算$dy/dt = t - 2y$的右端表达式
rhsDecay	12.4.1	计算$dy/dt = -\alpha y$的右端表达式
rhsPop2	12.5.1	计算两物种的猎食者-猎物模型的ODE组的右端表达式
rhsSmd	12.5.2	计算弹簧减震器系统的ODE组的右端表达式
rhsSteelHeat	12.5	计算热处理模拟模型ODE的右端表达式
rhsSys	12.5.1	计算例12.10中的ODE组的右端表达式

补充读物

很多数值计算方面的书都提供了ODE的数值解法。有关ODE积分算法分析的高层次的介绍，可以参考Shampine et al.[65]。要想知道更多的数学方面的介绍，见Stoer和Bulirsch [70]。在12.2.2节中关于离散误差的介绍就借用了这两处资料中的有关内容。

Polking [60]是本章材料的优秀补充，Polking使用MATLAB来推导微分方程的基础理论，他也用MATLAB来介绍ODE的数值求解算法。

727

Shampine和Reichelt [66]介绍了MATLAB中ODE求解程序的理论和实现。

习题

每个练习前圆括号中的数字表示练习的难度和完成练习所需要的工作量。

1. (1+) 在初始条件$T(t=0)=T_0$下求方程(12-5)的解析解。
2. (1) 在程序清单12-1的`odeEuler`函数中，初始条件如何存储在$y(1)$中。
3. * (1) 在$h=0.2$时，手动计算欧拉法的前3步来解下列ODE

$$\frac{\mathrm{d}y}{\mathrm{d}t}=\frac{1}{t+y+1},\qquad y(0)=0$$

4. (1) 在$0\leqslant t\leqslant 1$上$h=0.2$时使用`odeEuler`来求解上个练习中的ODE。
5. (2) 以`demoEuler.m`为参照写一个m文件，用欧拉法来解

$$\frac{\mathrm{d}y}{\mathrm{d}t}=\cos(t)\qquad y(0)=0,\ 0\leqslant t\leqslant 4\pi$$

   这需要一个简单的函数(如`rhscos.m`)来计算ODE的右端表达式。为什么不能使用内置的`cos`函数？画出$h=4\pi/100$和$h=4\pi/1000$时精确解与欧拉法求出的数值解的比较图。每个h值在$t=4\pi$的绝对误差是多少？在$0\leqslant t\leqslant 4\pi$上最大(绝对)误差是多少？
6. (2) 对上个练习加以扩展，作出在$t=4\pi$处误差的对数和$h=4\pi/n$时$0\leqslant t\leqslant 4\pi$上最大(绝对)误差的对数关系图形，其中$n=10,\ 10^2,\ 10^3,\ 10^4$。在同一个坐标系上画出两个误差图，讨论结果。随着$n$的增长，曲线的趋势会无限延续下去吗？
7. * (2+) (Stoer and Bulirsch [70]) 令$h=0.05$，使用欧拉法解

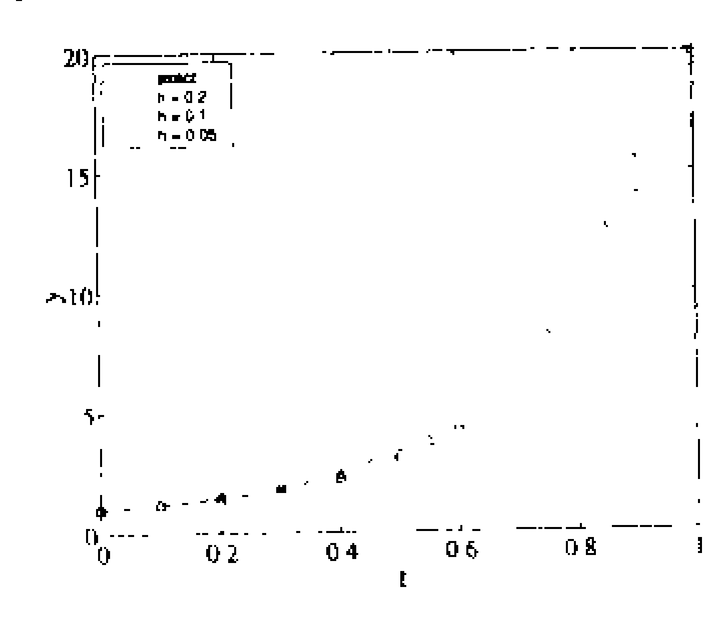
$$\frac{\mathrm{d}y}{\mathrm{d}t}=\sqrt{y}\qquad y(0)=0,\ 0\leqslant t\leqslant 2$$

   画出数值解和精确解的比较图。从图中可以看出`odeEuler`中存在一个错误吗？如果其中没有错误，你能解释这个特别的结果吗？使用`odeMidpt`和`odeRK4`重新求解。(提示：如果初始条件使用$y(0)=\varepsilon_m$来代替$y(0)=0$会怎样？)
8. (2) 用一个Heun法的MATLAB实现程序来对下列ODE积分

$$\frac{\mathrm{d}y}{\mathrm{d}t}=\frac{2(1-t)}{(0.05+(t-1)^2)^2}\qquad y(0)=1$$

728

   左边的图描述了使用一系列逐渐变小的步长来对ODE进行数值积分的结果。右边的表格列出了数值解和已知精确解间的最大误差。其MATLAB实现正确吗？用语言解释正确与否的原因。



h	$\max\lvert y_j-y(t_j)\rvert$
0.20	4.020
0.10	2.787
0.05	1.335

9. （2+）使用内置的flops函数来测量欧拉法和中点法在$0 \leqslant t \leqslant 2$区间上对方程（12-11）积分所需要的浮点操作次数，比较$h = 0.2$、$h = 0.1$、$h = 0.05$和$h = 0.025$时的结果。中点法会由于工作量变少而与欧拉法获得相当的精度吗？为你的答案提供定量化的阐述。

10. *（2+）参考odeEuler.m文件，写一个m文件来实现任意一阶ODE的Heun算法。使用此函数在$h = 0.2$、$h = 0.1$、$h = 0.05$和$h = 0.025$时解出方程（12-11），并将此程序的全局离散误差与Heun法全局离散误差的理论值进行比较。

11. （2）参考NMM工具箱中的compEMRK4函数，比较固定步长RK-4法和内置ode23及ode45在解方程（12-26）时的浮点操作次数和精度。

12. *（1+）对下列初值问题，验证给出的$y(t)$是所求的解。

(a) $\dfrac{dy}{dt} = \dfrac{t^2}{\alpha} - y;\quad y(0) = 1;\quad \Rightarrow\quad y(t) = \dfrac{1}{\alpha}[t^2 - 2t + 2 + (\alpha - 2)e^{-t}]$

(b) $\dfrac{dy}{dt} = 2t - 1 - \dfrac{1}{t+\alpha};\quad y(0) = -\ln(\alpha),\quad \Rightarrow\quad y(t) = t^2 - t - \ln(t+\alpha)$

(c) $\dfrac{dy}{dt} = \alpha t + \beta t y;\quad y(0) = 1;\quad \Rightarrow\quad y(t) = -\dfrac{1}{\beta}\left[\alpha - (\alpha+\beta)e^{\beta t^2/2}\right]$

729

(d) $\dfrac{dy}{dt} = \dfrac{2(\beta - t)}{\left[\alpha + (t-\beta)^2\right]^2};\quad y(0) = \dfrac{1}{\alpha + \beta^2};\quad \Rightarrow\quad y(t) = \dfrac{1}{\alpha + (t-\beta)^2}$

(e) $\dfrac{dy}{dt} = \alpha y - \beta y^2\quad y(0) = y_0\quad \Rightarrow\quad y(t) = \dfrac{\alpha y_0}{\beta y_0 + (\alpha - \beta y_0)\exp(-\alpha t)}$

13. （1+）在$0 \leqslant t \leqslant t_n$上使用欧拉法解习题12的初值问题，采用一系列逐渐减小的h。画出精确解和数值解的图形，并报告t离散值上的最大误差。使用下列α, β, y_0和t_n值。

(a) $\alpha = 3,\ y_0 = 1,\ t_n = 2$

(b) $\alpha = 1/20,\ t_n = 2$

(c) $\alpha = 1,\ \beta = -2,\ y_0 = 1,\ t_n = 4$

(d) $\alpha = 1/20,\ \beta = 1,\ t_n = 2$

(e) $\alpha = 2,\ \beta = 1/50,\ y_0 = 1,\ t_n = 8$

14. （1+）重复习题13，使用中点法来求解各方程。

15. *（1+）重复习题13，使用内置ode23法和ode45法来解各方程。不要采用一系列逐渐减小的h（为什么？），使用缺省的收敛参数，比较ode23法和ode45法得到的解。

16. （2+）使用odeRK4v代替ode45求解例12.9中的热处理模型。此函数包含在NMM工具箱的ode目录下，应用它可避免改动rhsSteelHeat函数。要使T_{max}与缺省参数下的ode45得到的值相差小于1℃，h需要为多少？

17. （2+）使用内置的ode45解习题7中的ODE。比较由$y(0)= 0$（确定值）和$y(0)=\varepsilon_m$得到的数值解和精确解，解释所求解不同的原因。

18. （1+）对于以下输入运行demoPredPrey程序

(a) $p_1(0)=5000$, $p_2(0)=10$

(b) $p_1(0)=500$, $p_2(0)=10$

(c) $p_1(0)=50$, $p_2(0)=10$

(d) $p_1(0)=5000$, $p_2(0)=300$

(e) $p_1(0)=50000$, $p_2(0)=300$

在所有这些条件中，哪些数量特性与种群数量随时间变化的特性相一致？

19. (1+) 使用odeRK4sysv函数解例12.11中的猎食者–猎物模型ODE。假设初始条件和α、δ值都相同，对$h = 0.5$，0.1，0.05的情况分别进行模拟。固定步长法得到的种群数量变化情况与使用ode45得到的相同吗？

20. (3) 根据odeEuler函数，写一个odeEulerSys函数来对ODE组积分，并用例12.11中的猎食者–猎物模型检验所写的程序。此过程中，需要将α和β传给计算ODE右端表达式的m文件。推荐使用varargin工具（见odeRK4sysv中的代码），另外，也可以用全局变量。使用相同的初始条件和α、δ值，对h= 0.05、0.02、0.01的情况进行模拟。欧拉法与ode45得到的种群数量变化情况有什么定性的不同？ 730

21. (3) 使用中点法对方程组积分，重复上个练习。

22. (2) Rössler方程组形式如下（见Moon [57]问题5-9）

$$\dot{x} = -y - z, \quad \dot{y} = x + ay, \quad \dot{z} = b + z(x - c)$$

已知$a = 0.398$、$b = 2$、$c = 4$且$0 \leqslant t \leqslant 100$，求此方程组的数值解。首先用随机初始条件（y0 = rand(1, 3)）将这个计算过程重复四次，然后使用内置的plot3函数产生解的轨迹（$x(t)$, $y(t)$, $z(t)$）的三维图，再用subplot(2, 2, i)，i = 1:4将四次运行的图放在同一个图形窗口中。初始条件对轨迹有很大的影响吗？

23. (2+) 使用带Refine选项的demoODE45opts函数可以对ode45函数进行实验。例如，下面的语句可以将相同容差的解重叠在一张图上

```
>> demoODE45opts(1e-3,1e-6,1)
>> hold on, demoODE45opts(1e-3,1e-6,4); hold off
```

除了由插值产生的额外点，两个解曲线看起来完全一样。两次运行demoODE45opts得出的Max error值不同。如果两个解真的相同，那么Max error也就应该相同，解释为什么Max error会有差异。给出demoODE45opts代码的修改建议，使它产生的Max error与nref无关。（提示：所作的改变只影响用来计算emax的值，不会影响传给ode45的参数）。

24. (3) 例12.12中物体的位置x和速度$\dot{x}$的精确解为

$$x_e(t) = \frac{a_0}{\omega_n^2}\left[1 - e^{-\zeta\omega_n t}(\cos\omega_d t - \gamma\sin\omega_d t)\right]$$

$$\dot{x}_e(t) = \frac{a_0}{\omega_n}\sqrt{1-\zeta^2}\, e^{-\zeta\omega_n t}\sin\omega_d t$$

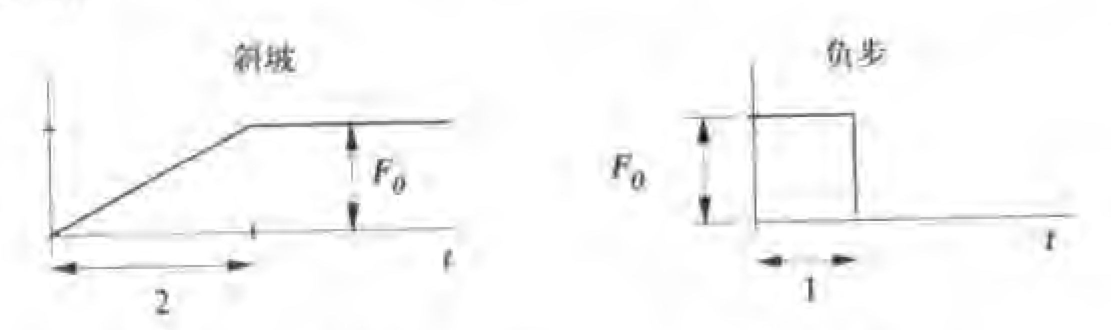
其中

$$\omega_d = \omega_n\sqrt{1-\zeta^2} \qquad \gamma = \frac{\zeta}{1-\zeta^2}$$

修改demoSmd函数，向图中添加位移和速度的精确解，并打印数值求解过程使用的时间步总数。为此函数添加语句，用来计算并打印$\|x - x_e\|_\infty$和$\|\dot{x} - \dot{x}_e\|_\infty$，其中$x$和$\dot{x}$是ode45返回的数值解。再添加一些语句，从数值解中求出x_{max}、$t(x_{max})$、v_{max}和$t(v_{max})$，其中x_{max}是物体的最大位移，v_{max}是物体的最大速度。（提示：[xmax, ixmax] = max(x(:, 1))。） 731

使用缺省的收敛容差，在`tstop=` 0.2s时运行修改后的`demoSmd`函数。将`RelTol`和`AbsTol`减小为缺省值的1/1000，再次运行`demoSmd`函数（运行8到10次就能得到清晰的趋势）。将结果总结在一张表中，说明 $\|x-x_e\|_\infty$、$\|\dot{x}-\dot{x}_e\|_\infty$、$x_{max}$，$t$（$x_{max}$），$v_{max}$和$t$（$v_{max}$）是如何依赖于收敛容差的。`RelTol`和`AbsTol`对数值解的影响大吗？评价精确解和数值解的比较图。

25. （2+）修改`rhsSmd`函数（见程序清单12-13），实现例12.12中二阶弹簧减震器系统的下列受力函数。



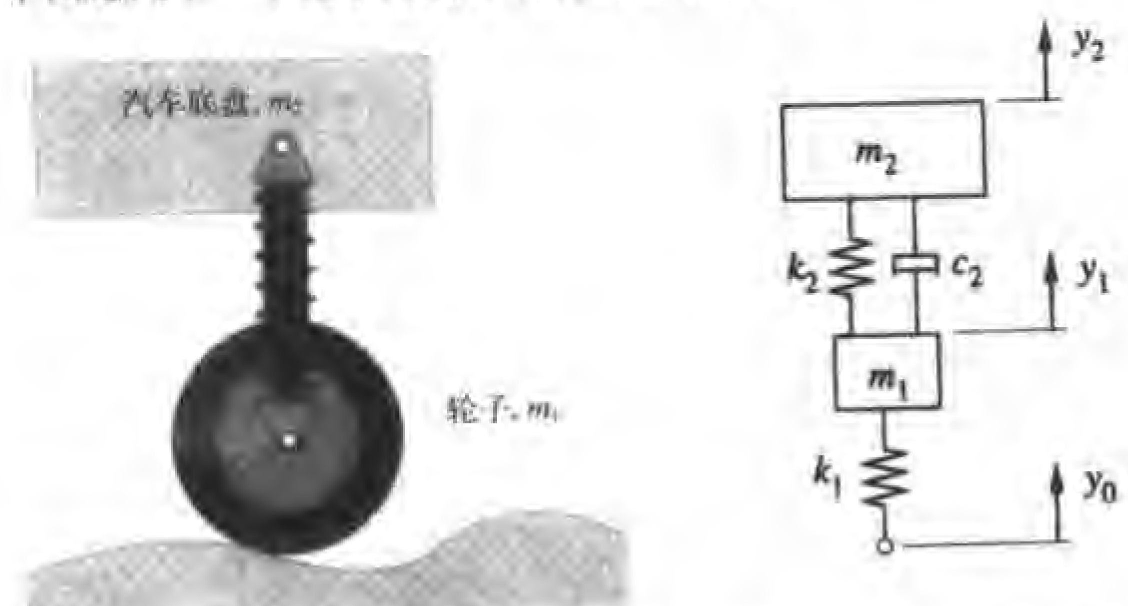
*(a) 施加一个正弦曲线力F（t）= $F_0\sin$（ωt），其中$\omega \neq \omega_n$。

(b) 从0到2秒施加一个线性增加的力，然后恒定在F_0。

(c) 施加恒定力F_0持续1秒，然后撤去力（负步，negative step）。

设F_0= 1，比较系统的反应。（提示：对三个条件分别写一个独立的m文件函数计算其$F(t)$。运行时选择某一个$F(t)$，但不能每次都编辑`rhsSmd`。实现的方法之一就是将所选$F(t)$的m文件名由`ode45`传给修改后的`rhsSmd`函数，然后由内置的`feval`函数调用$F(t)$函数。其中，$F(t)$的m文件函数名也应该作为输入传给`demoSmd`函数。）

26. （3－）下图是汽车一个轮子的缓冲系统的简易模型：



732

系统的输入是根据不同地形随时间变化的位移y_0（t）。震动缓冲器有一个弹性系数k_2和减震系数c_2，轮胎中的减震不予考虑（没有c_1项）。

对轮子和底盘应用牛顿运动定律和受力平衡，得到如下方程组：

$$m_1\ddot{y}_1 + c_2(\dot{y}_1 - \dot{y}_2) + k_2(y_1 - y_2) + k_1 y_1 = k_1 y_0(t)$$
$$m_2\ddot{y}_2 - k_2(y_1 - y_2) - c_2(\dot{y}_1 - \dot{y}_2) = 0$$

(a) 将这两个二阶方程转变成等价的一阶方程组（需要几个一阶方程？）。

(b) 使用合适的ODE积分程序解此方程组，其中m_1 = 110kg，k_1 = 136 N/m，m_2 = 1900kg，k_2 = 16 N/m，c_2 = 176 N s/m，受力函数为y_0（t）= 0.05sin（$3\pi t$）。

(c) 将c_2减小为原来的1/5，重复以上计算。请描述由于c_2的改变所引起的方程特性的变化情况。

27. *（2+）Duffing方程

$$\frac{d^2x}{dt^2}+kx+x^3=B\cos t$$

描述了非线性感应电路的混沌动力学（例见Moon[57]，第6章）。将此方程转变成两个一阶ODE组并求解，其中$k = 0.1$、$B = 12$、$0 \leqslant t \leqslant 100$。画出$dx/dt$和$x$的关系图，画出Poincaré图。

733~740

附录A 特征值和特征方程组

毫无疑问，读者之前已经遇到特征值问题了。例如，在求解齐次线性微分方程组时，可接受的解的形式就是由特征值问题所决定的，其基解(building block solution)就称为特征函数。要解微分方程，必须找到特征值-特征函数对（eigenvalue-eigenfunction pair），它们一般有无穷多个。但是本章中考虑的矩阵（或代数）特征值问题却有所不同。矩阵特征值问题一般出现在物理系统的离散模型中，这些离散模型的自由度有限，所得特征值问题也只涉及到求有限个特征值-特征向量对。特征向量是离散的——它们包含有限个元素——与本书其他地方考虑的向量一样。

矩阵特征值问题是数值分析中很大也是很重要的一个部分。本附录将简单介绍特征值和特征向量，并示范如何使用内置eig函数来求特征值和特征向量。

741

在许多的应用问题中计算一个矩阵的特征值很重要。

- 在数值分析中，对于一个涉及矩阵的迭代序列，其收敛性决定于迭代矩阵的特征值大小。
- 在动态系统中，特征值能指明系统是振荡的、稳定的（衰减振荡）或是不稳定的（增强振荡）。
- 在振荡系统中，微分方程（对于一个连续模型）或有限元模型的系数矩阵的特征值与系统的固有频率直接相关。
- 在回归分析中，用相关矩阵的特征值来选择新的预测变量或者主成分（principle component），它是原始预测变量的线性组合。

内置的eig函数在这些应用中非常有用。如果只关系到有限的几个离散特征值，就可以用eig函数来进行计算。

A.1 特征向量对本身的映射

矩阵与向量的乘积$y = Ax$可以解释为矩阵A对向量x的旋转和伸缩。所得向量y显然决定于A和x。但是，有些与A有关的特殊向量却不能被A旋转，只能通过A的作用伸长或缩短，这些向量就是A的特征向量。

考虑程序清单A-1中的iterMult函数。已知矩阵A和与其匹配的向量x，使用iterMult函数可作以下变换：

$$
\begin{aligned}
u_{(0)} &= x \\
u_{(1)} &= Au_{(0)} \\
u_{(2)} &= Au_{(1)} = AAu_{(0)} \\
&\vdots \\
u_{(k)} &= Au_{(k-1)} = \underbrace{AA\ldots A}_{k\text{次}}x
\end{aligned}
$$

圆括号中的下标强调$u_{(0)}, u_{(1)}, \ldots, u_{(k)}$是$k$个向量的序列，而不是向量$u$的第$1, \ldots, m$个元素。除

因为每一步中向量$u_{(k)}$都可以由其L_∞范数来衡量，所以就可以使用iterMult函数计算此向量序列。函数iterMult的特性非常有趣。对很多矩阵来说，向量序列

$$x\,,\ Ax\,,\ A^2x\,,\ A^3x\,,\ \ldots$$

742

收敛于一个向量，该向量依赖于A而不是初始向量x。

程序清单A-1 函数iterMult重复地用向量A乘以向量x_0，并对结果进行了衡量

```
function [u,lambda] = iterMult(A,x,nit)
% iterMult  Iterated multiplication of a vector by a matrix: u = A*A*...*A*x
%
% Synopsis:  u            = iterMult(A,x,nit)
%            [u,lambda] = iterMult(A,x,nit)
%
% Input:     A = an n by n matrix
%            x0 = n by 1 (column) vector to start the iterations
%            nit = number of iterations
%
% Output:    u = A*A* ... A*x, result of m multiplications of A on x
%                scaled by the max norm of the result at each step
%                The max norm of u is printed at each step
%            lambda = (optional) infinity norm of the scaled u
u = x;
fprintf('  k    norm(u,inf)\n');
for k=1:nit
  u = A*u;
  r = norm(u,inf);
  u = u/r;
  fprintf('%3d     %f\n',k,r);
end
if nargout==2, lambda = r; end
```

下面是对(5,2,-1)决定的三对角矩阵进行50次iterMult迭代的结果（经过编辑）：

```
>> A = tridiag(5,2,-1);    % tridiag from linalg directory of NMM toolbox
>> x = rand(5,1);
>> u = iterMult(A,x,50)
  k     norm(y,inf)
  1      1.190089
  2      2.214240
  3      3.650440
  :        :
 49      3.732051
 50      3.732051

 u =
   -0.5000
    0.8660
   -1.0000
    0.8660
   -0.5000
 lambda =
     3.7321
```

743

你也可以产生另一个随机的向量x或选择特殊向量如$x = [1,0,0,0]^T$及$x = [0,1,0,0]^T$来进行测试，以检验本例的结果是否依赖于初始向量。

函数iterMult的输出说明A^ku的L_∞范数收敛于一个常数。再额外手工计算几次迭代，可以发现A^ku也变成了常数。

```
>> w = A*u;
>> lambda = norm(w,inf)
lambda =
    3.7321

>> unew = w/lambda;
>> disp([w  u  unew])
   -1.8660   -0.5000   -0.5000
    3.2320    0.8660    0.8660
   -3.7321   -1.0000   -1.0000
    3.2321    0.8660    0.8660
   -1.8661   -0.5000   -0.5000
```

不出所料，unew与前面迭代的u（几乎）一样。由于iterMult的舍入和收敛性质，unew和u不会精确相等（它们有多接近？）。

前面的计算可总结如下。已知矩阵A和一个随机的初始向量u_0，向量序列$u(k) = Au_{k-1}$，$k = 1,2,\ldots$会收敛于$\upsilon = \lim_{k\to\infty} u_{(k)}$。向量$\upsilon$有一个特殊属性，$A\upsilon$的值相当于$\upsilon$和一个标量的乘积，换言之，

$$A\upsilon=\lambda\upsilon \tag{A-1}$$

一般地，方程(A-1)定义了一个特殊的标量-向量对，其中λ是特征值（*eigenvalue*），u是A的特征向量（*eigenvector*）。既然乘积$A\upsilon$等于υ乘以一个标量，那么可以推断A不能将其特征向量旋转。

A的特征向量和特征值都是A的特征。函数iterMult中的迭代只是产生(λ,υ)对的一种简单方法　这种方法被称为乘幂法(power method)，将在随后一节进行介绍。对任意的矩阵A计算其特征值和特征向量是一个具体的数值任务，这需要比乘幂法更复杂的方法。参考[12、78]中关于矩阵特征值计算的介绍。

744

A.2　数学预备知识

本节对矩阵特征值问题中的一些数学术语作简要总结。

A.2.1　特征多项式

方程（A-1）定义了矩阵的特征值问题。在手工计算和理论分析中，有一种很有用的等价形式。方程两边同时减去$\lambda\upsilon$，得

$$(A-\lambda I)\upsilon=0$$

其中I是单位矩阵。解此方程需要$\upsilon=0$（平凡解）或

$$\det(A-\lambda I)=0 \tag{A-2}$$

其中det(•)是其矩阵参数的行列式（*determinant*）。对$n\times n$的矩阵A计算方程（A-2），可得到关于λ的n阶多项式，称为A的特征多项式（*characteristic polynomial*）：

$$\det(\boldsymbol{A}-\lambda\boldsymbol{I})=p(\lambda)=\lambda^{n}+\alpha_{n-1}\lambda^{n-1}+\alpha_{n-2}\lambda^{n-2}+\cdots+\alpha_{0} \tag{A-3}$$

此多项式的根也就是矩阵$\boldsymbol{A}$的特征值。这表明求矩阵特征值问题可以转变成多项式的求根问题。虽然这在理论上是可行的，但是由于存在重复根且多项式求根问题是病态的，此方法也很复杂。一般地，通过求$p(\lambda)$的根来求$\boldsymbol{A}$的特征值并不是一个好办法。

例A.1 矩阵的特征多项式

计算

$$\boldsymbol{A}=\begin{bmatrix}3 & -2 & 1\\ -2 & 4 & -1\\ 1 & -1 & 5\end{bmatrix}$$

的特征多项式。想要得到的结果是通过直接计算

$$\det(\boldsymbol{A}-\lambda\boldsymbol{I})=\begin{vmatrix}(3-\lambda) & -2 & 1\\ -2 & (4-\lambda) & -1\\ 1 & -1 & (5-\lambda)\end{vmatrix}$$

得出

$$\det(\boldsymbol{A}-\lambda\boldsymbol{I})=\lambda^{3}-12\lambda^{2}+41\lambda-37 \tag{A-4}$$

745

内置的poly函数可以计算一个矩阵的特征多项式：

```
>> A = [3 -2  1; -2 4 -1; 1 -1 5];
>> p = poly(A)
p =
    1.0000   -12.0000    41.0000   -37.0000
```

向量$\boldsymbol{p}$的元素依λ的降幂排列，它们与方程(A-4)一致。

A.2.2 伴随矩阵

在进行一般的求矩阵特征值问题之前，我们再作一次简略的回顾，重新考虑一下多项式求根的问题。虽然使用求根法来求矩阵的特征值并不是一个好办法，但是把问题反过来看，也就是说通过求特征值问题来得到多项式的根，却是一个很好的办法（参照6.7节）。

如果$\boldsymbol{A}$的特征多项式由方程(A-3)给出，那么$\boldsymbol{A}$的伴随矩阵就是

$$\boldsymbol{C}=\begin{pmatrix}-\alpha_{n-1} & -\alpha_{n-2} & -\alpha_{k-1} & \cdots & -\alpha_{1} & -\alpha_{0}\\ 1 & 0 & 0 & \cdots & 0 & 0\\ 0 & 1 & 0 & \cdots & 0 & 0\\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots\\ 0 & 0 & \cdots & 1 & 0 & 0\\ 0 & 0 & \cdots & 0 & 1 & 0\end{pmatrix}$$

通过直接求解（可用3×3矩阵验证）可得，$\boldsymbol{C}$的特征多项式和方程(A-3)相同。

A.2.3 相关结论（Eigenfact）

下面的事实在解特征值问题时很有用（例见[12、58、71、78]中关于这些观点的更详细推导）：

- 一个矩阵的特征值可以重复，例如

$$A=\begin{bmatrix}1 & 1\\ -1 & 3\end{bmatrix}$$

746

就有重复的特征值$\lambda=2$。

- 若A为$n\times n$矩阵，它至少有一个不重复的特征值（distinct eigenvalue），最多有n个特征向量。
- 任何方阵的对角线元素之和等于此矩阵的所有特征值之和

$$\sum_{i=1}^{n}a_{ii}=\sum_{i=1}^{n}\lambda_i$$

其中λ_i是A的特征值。一个矩阵的对角线元素之和也称为此矩阵的迹（*trace*）（即$\text{trace}(A)=\sum_{i=1}^{n}a_{ii}$）。

- 任何矩阵的所有特征值的乘积等于矩阵行列式的值：

$$\prod_{i=1}^{n}a_{ii}=\det(A)$$

- 若A为非对称实矩阵，其特征值可能为实数，也可能为复数。复特征值（complex eigenvalue）按共轭形式成对出现。例如，

$$B=\begin{bmatrix}1 & 1\\ -2 & 3\end{bmatrix}$$

就有复特征值$\lambda=2\pm i$。

- 若A是对称的，其特征值为实数。
- 若A的特征值全为正实数，那么A就是正定的，且对任意含n个实数的列向量x有

$$\psi(x)=x'Ax>0$$

ψ称为纯二次型（*pure quadratic form*）（说"二次"是因为x以二次项的形式出现（例如，x_1^2、$x_1x_2,\ldots$），说"纯"是因为$\psi(x)$的表达式中只有x的二次项出现（例如，没有$x_1x_2,\ldots$或常数项））。

- 若一个矩阵有n个不同的特征值，那么它就有n个线性无关的特征向量。
- 若$n\times n$矩阵A有n个不同的特征值，那么A就可以分解成

747

$$A=X\Lambda X^{-1}\tag{A-5}$$

其中

$$\Lambda=\begin{bmatrix}\lambda_1 & 0 & 0 & 0\\ 0 & \lambda_2 & & \vdots\\ \vdots & & \ddots & 0\\ 0 & \cdots & 0 & \lambda_n\end{bmatrix}$$

且X的各列为A的特征向量。方程(A-5)有时称为A的特征值分解（*eigenvalue decomposition*）或谱分解（*spectral decomposition*）。既然方程(A-5)中的分解要求A有n个不重复的特征值，那么就一定要记住，对一般的矩阵A而言特征值分解可能不存在。若A可以被分解成方程(A-5)的形式，就说A是可对角化(diagonalizable)的。

- 矩阵的特征向量乘以标量后构成的向量仍然是矩阵的特征向量。为证明之，令$w=v/\sigma$，其中σ是任意标量。将$v=\sigma w$代入方程(A-1)中，得

$$A(\sigma w) = \lambda(\sigma w)$$
$$\sigma A w = \sigma \lambda w$$
$$A w = \lambda w$$

因此v和w都是和特征值λ有关的特征向量。

前面的事实列表至少在两方面很有帮助。当要处理的实际问题可归结为矩阵特征值问题的时候，由矩阵的属性可以推断出解的属性。例如，如果矩阵特征值问题是由微分方程得来，而且有关矩阵也是对称的，那么不需要深入分析我们就可以断定方程组不会振荡，因为振荡方程组对应矩阵的特征值是按共轭形式成对出现的复数。另一方面，在用处理数值方法求特征值和（可能的）特征向量时，如果遇到不对称的矩阵，就意味着不能再使用只适用于对称矩阵的诀窍。

在本书中，特征相关结论列表应该看成一种便捷的参考，它们不能替代对矩阵特征值问题的全面系统的讨论。

A.3 乘幂法

程序清单A-1中的`iterMult`函数实现了所谓的乘幂法，它用来计算一个矩阵的最大特征值（largest eigenvalue）或主特征值（dominant eigenvalue，也称优势特征值）。乘幂法主要用于只需要最大特征值和最大特征值与其他特征值不同的情况，它在实际使用时是非常有效的。它很适合用在大型稀疏矩阵中，因为它的大量的计算工作都花费在计算矩阵-向量的乘积上。如果（大型且稀疏的）矩阵以稀疏矩阵的格式存储（见附录B），那么就可以有效地计算矩阵-向量的乘积。乘幂法放在本节来推导是因为它是一个简单的算法，它显示出了矩阵特征值的一些属性，乘幂法与解$Ax = b$的定长迭代法（stationary iterative method）的收敛性研究也很有关系。对任意的矩阵来说，乘幂法可能失败，或者其收敛速度可能非常慢。 748

A.3.1 乘幂迭代

假设矩阵A有n个线性无关的特征向量$v_{(1)}$、$v_{(2)}$、…、$v_{(n)}$。记住，v的圆括号下标表示一系列向量，而不是向量的某些元素。根据Watkins[78,Corollary 4.2.11]，所有矩阵中，大部分都适用于有线性无关特征向量的假设。乘幂法在很多实际问题中很有用，我们将在这个观察结果和已知信息的基础上继续深入探讨。

为方便起见，令A的n个特征值按绝对值大小排列如下：

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n| \tag{A-6}$$

既然A的n个特征向量线性无关，那它们就形成了n维空间的基，任何n维向量x都可以写成：

$$x = \alpha_1 v_{(1)} + \alpha_2 v_{(2)} + \cdots + \alpha_n v_{(n)}$$

其中α_i是标量系数。在当前应用中，只要$\alpha_1 \neq 0$，α_i的大小就并不重要。已知x的这种表示法，乘积Ax可表示为

$$Ax = A\left[\alpha_1 v_{(1)} + \alpha_2 v_{(2)} + \cdots + \alpha_n v_{(n)}\right] = \alpha_1 A v_{(1)} + \alpha_2 A v_{(2)} + \cdots + \alpha_n A v_{(n)}$$

由方程(A-1)可知，每个$Av_{(i)}$项可以被$\lambda_i v_{(i)}$替代，其中$(\lambda_i, v_{(i)})$是一个特征值-特征向量对。于是

$$Ax = \alpha_1\lambda_1 v_{(1)} + \alpha_2\lambda_2 v_{(2)} + \cdots + \alpha_n\lambda_n v_{(n)}$$

左乘以A、并用$(\lambda, v_{(i)})$来代替$Av_{(i)}$，得

749

$$A\,Ax = A^2x = \alpha_1\lambda_1^2 v_{(1)} + \alpha_2\lambda_2^2 v_{(2)} + \cdots + \alpha_n\lambda_n^2 v_{(n)}$$

而且，一般地有

$$A^k x = \alpha_1\lambda_1^k v_{(1)} + \alpha_2\lambda_2^k v_{(2)} + \cdots + \alpha_n\lambda_n^k v_{(n)}$$

其中 $A^k = \underbrace{A\ A\ldots A}_{k次}$。

从以上表达式的右端提取因子$\lambda^k{}_1$，得

$$A^k x = \lambda_1^k\left[\alpha_1 v_{(1)} + \alpha_2\left(\frac{\lambda_2}{\lambda_1}\right)^k v_{(2)} + \cdots + \alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^k v_{(n)}\right] \tag{A-7}$$

观察以上方程可知乘幂法是如何工作的。由方程（A-6），可知对所有i>1有(λ_i/λ_1)<1。因此，方程（A-7）的方括号中的所有项将随着k的增加而趋于零，只有$\alpha_1 v_{(1)}$项除外：

$$A^k x \to \lambda_1^k\alpha_1 v_{(1)}\ 当\ k \to \infty \tag{A-8}$$

此外，当k增加时，方程（A-7）的右端接近于一个与特征向量$v_{(1)}$同方向的向量。实际上，因为特征向量只由一个任意倍增的常数因子决定，$\lambda^k{}_1\alpha_1 v_{(1)}$就是$\lambda_1$对应的特征向量$v_{(1)}$。

因此，对任意x和足够大的k计算$A^k x$将会产生A的主特征向量。这在严格的算术意义上是正确的（只要A的主特征值存在），但对较大的n或λ_1，$A^k x$中的系数大小就会很快增长以致发生上溢。上溢可通过在每一次迭代中缩放乘积$A^k x$来避免。其中一个合适的缩放因子就是$A^k x$本身的L_∞范数，因为它产生了最大元素为1的向量[⊖]。

乘幂法的基本计算可总结为

$$u_{(k)} = \frac{A^k x_0}{\|A^k x_0\|_\infty}, \quad k = 1,2,\cdots, \tag{A-9}$$

其中x_0是一个任意的初始向量。对足够大的k来说，$u_{(k)}$的值变为（由方程(A-8)）：

750

$$u_{(k)} = \frac{A^k x_0}{\|A^k x_0\|_\infty} = \frac{\lambda_1^k\alpha_1 v_{(1)}}{\|\lambda_1^k\alpha_1 v_{(1)}\|_\infty} = \frac{\lambda_1^k\alpha_1 v_{(1)}}{\lambda_1^k\alpha_1\|v_{(1)}\|_\infty} = \frac{v_{(1)}}{\|v_{(1)}\|_\infty}\ 当\ k \to \infty$$

也就是说，乘幂迭代常数对应于主特征值的规格化特征向量，实现乘幂法的高效程序在每步迭代时不重新计算$A^k x_0$，而是由公式

$$u_{(k)} = \frac{Au_{(k-1)}}{\|Au_{(k-1)}\|_\infty}\ 和\ \lim_{k\to\infty} u_{(k)} = v_{(1)}$$

来递归地计算$u_{(k)}$向量序列。其中$v_{(1)}$是A的主特征向量。

对应于主特征向量的特征值可以使用乘幂法轻易得到。定义一个临时向量$w = Au_{(k-1)}$，然后在收敛处$w = Au_{(k-1)} = \lambda_1 u_{(k-1)}$，所以

⊖ 也可以使用L_2范数，这样的话结果就是乘幂法产生的向量序列长度为1，而不是其最大元素为1。虽然两种范数都可以使用，但使用L_∞所作的浮点计算量较少。

$$\frac{\|w\|}{\|u_{(k-1)}\|}=\lambda_1$$

但是，因为$\|u_{(k-1)}\|_\infty=1$（见方程式（A-9）），故有

$$\lambda_1=\|w\|$$

将前面的想法组织成一个连续的处理过程，可得到乘幂法的算法如下。

算法A-1 乘幂迭代

```
initialize u(0)
for k=1,2,...
  w=Au(k-1)
  λ=||w||∞
  u(k)=(1/λ)w
end
```

在算法A-1中，w是一个临时向量。既然对迭代的历史数据没有兴趣，那么中间的$u_{(k)}$就不需要存储，可以通过覆盖向量u来避免产生临时的w向量。注意，乘幂算法中，$u_{(k)}$只有在收敛时才是矩阵A的特征向量。

程序清单A-1中的`iterMult`函数就实现了乘幂迭代。NMM工具箱中的函数`powerit`（此处未列出）是`iterMult`函数的修改版，它更适合特征值计算。

A.3.2 反乘幂迭代

反乘幂法（*inverse-power method*、逆幂法）可适用于求解矩阵的最小特征值。将方程(A-1)两边左乘A^{-1}，得

751

$$A^{-1}Av=\lambda A^{-1}v$$

$$\lambda^{-1}v=A^{-1}v$$

如果我们定义新矩阵$B=A^{-1}$和新标量$\mu=\lambda^{-1}$，那么上面的方程式就等价于

$$Bv=\mu v$$

因此，如果(λ,v)为A的特征值-特征向量对，那么(λ^{-1},v)就是A^{-1}的特征值-特征向量对。对$B=A^{-1}$应用乘幂迭代（如果算法收敛）就可得A的最小特征值。

乘幂法的简单实现可以直接计算$B=A^{-1}$，并对B使用算法A-1。相反地，考虑算法A-1中的矩阵相乘步骤$w=Au_{(k-1)}$。在反乘幂法中矩阵A由$B=A^{-1}$代替，则语句$w=Bu_{(k-1)}=A^{-1}u_{(k-1)}$就等价于给定$u_{(k-1)}$的关于$w$的解

$$Aw=u_{(k-1)}$$

由于8.1.2节中讨论的原因，相对于求解$w=A^{-1}u_{(k-1)}$，我们更倾向于求解上面的方程组。此外，既然A在迭代中不变，那么A的LU分解就可以在迭代开始时进行一次。这样，表达式“求解$Aw=u_{(k-1)}$”就可以解释成在每次迭代中应用两个三角求解序列（先向前代入然后向后代入）。下面的算法明确了这个过程。

算法A-2　反乘幂迭代

initialize $u_{(0)}$
factor: $A=LU$
for $k=1,2,\ldots$
　solve: $L\omega=u_{(k)}$
　solve: $Uu_{(k+1)}=w$
　$\mu=\|w\|_\infty$
　$u_{(k)}=(1/\mu)w$
end
$\lambda=1/\mu$

实际上，反迭代法是通过变换(移位)特征值使最小特征值接近于零这一简单操作来加快收敛速度的（例见Watkins[78]）。在NMM工具箱的eigen目录下的poweritt和poweritInv函数分别实现了变换乘幂迭代和变换反乘幂迭代。

752

A.4　用来计算特征值的内置函数

乘幂法在只需要计算矩阵的少数几个实数特征值且这些特征值的绝对值不同的情况下才有用。对一般的矩阵，这两个条件很难满足，因此需要使用更复杂的方法来求特征值。

QR算法就是一个计算矩阵所有特征值的强大的方法。QR算法将给定矩阵迭代转变为另一个具有相同特征值的矩阵，在特征值相同的意义上，这两个矩阵是等价的。特征值和（如果希望计算）特征向量就可以有效地从转变后的矩阵中求出。虽然QR算法使用矩阵的QR分解（参见9.2.3节）来计算特征值，但求解特征值问题比单纯地求QR分解更困难和棘手。参考[12、15、32、76、78]中关于QR算法如何用于解代数特征值问题的讨论。

A.4.1　eig函数

函数eig使用QR算法可轻易地求矩阵的所有特征值和所有特征向量。在MATLAB第5版及更新的版本中，输入矩阵就可能是稠密（密集）的，也可能是稀疏的。此函数有两种调用形式：

```
lam = eig(A)
[V,L] = eig(A)
```

其中A是方阵，lam是由特征值构成的列向量，L是对角矩阵，其对角线元素为A的特征值（即lam = diag(L)），V是由A的特征向量组成的矩阵。在MATLAB命令提示符状态下键入help eig可得到更多关于使用eig的信息。

例A.2　eig函数的应用

内置的eig函数能轻易地求一个三对角矩阵的特征值和特征向量。例如，考虑下列语句：

753

```
>> A = tridiag(5,2,-1)     % tridiag is an NMM toolbox function
A =
     2    -1     0     0     0
    -1     2    -1     0     0
     0    -1     2    -1     0
     0     0    -1     2    -1
```

```
     0     0     0    -1     2
>> lam = eig(A)
lam =
    3.0000
    1.0000
    2.0000
    3.7321
    0.2679
```

注意，最大的特征值与A-1中使用乘幂迭代求出的结果相同。A的特征值和特征向量的全集为

```
>> [V,L] = eig(A)
V =
   -0.5000    0.5000    0.5774    0.2887    0.2887
   -0.5000   -0.5000    0.0000    0.5000   -0.5000
         0   -0.0000   -0.5774    0.5774    0.5774
    0.5000    0.5000    0.0000    0.5000   -0.5000
    0.5000   -0.5000    0.5774    0.2887    0.2887
L =
    1.0000         0         0         0         0
         0    3.0000         0         0         0
         0         0    2.0000         0         0
         0         0         0    0.2679         0
         0         0         0         0    3.7321
```

记住特征值和特征向量是成对出现的这一性质特别重要。相应地，A的第k个特征值是`L(k,k)`，对应的特征向量是`V(:,k)`。因此，主特征值-特征向量对可由以下语句得到：

```
>> [lmax,imax] = max(abs(diag(L)))
lmax =
    3.7321
imax =
     5

>> V(:,imax)
ans =
    0.2887
   -0.5000
    0.5774
   -0.5000
    0.2887
```

754

NMM工具箱中的`eigSort`函数（此处未列出）求出一个矩阵的特征值和特征向量，并返回以λ的升序或降序排列的V和L矩阵。

A.4.2 `eigs`函数

函数`eigs`可以求出一个矩阵的有限个特征值。当系数矩阵为稀疏矩阵（参见附录B）且只需要求解少数几个特征值时，使用此函数比使用`eig`函数效率更高。

A.5 奇异值分解

任何$m \times n$的实数矩阵A（$m \geqslant n$）可分解为三个矩阵的乘积

$$A = U\Sigma V^T \tag{A-10}$$

其中U是正交的$m \times M$矩阵，Σ是$m \times n$的对角矩阵，V是正交的$n \times n$矩阵。这个有点令人生畏的表达式提供了检查A的基础属性（underlying property）的方法。方程(A-10)就是A的奇异值分解。

矩阵Σ的元素就是A的奇异值(singular value)。Σ可以形象地表示为

$$\Sigma = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ 0 & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{A-11}$$

其中$\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_n$。在严格的算术意义上，$A$的非零奇异值的个数等于$A$的秩。在浮点算术意义上，若一个矩阵最小的奇异值$\sigma_n$远小于$\sigma_1$，那么此矩阵就近似于奇异的。实际上，一个方阵（$m = n$）的$L_2$范数的条件数为

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n} \quad \text{其中}A\text{是}n \times n\text{方阵} \tag{A-12}$$

根据约定，$\kappa_2=\sigma_1/\sigma_n$也适用于矩形矩阵（即$m>n$）。

在A是对称矩阵的特殊情况下，SVD和特征值分解（参见A.2.3节）是相关的。特别地，当A对称时A的奇异值就是其特征值的绝对值。一般地，不管A是否是方阵或是否对称，当$m \geqslant n$时，都有A的奇异值是A^TA的特征值，当$m<n$时A的奇异值是AA^T的特征值。不要只关心特殊情况下SVD和特征值问题的相应性，考虑这些分解的差异更有用。

755

方程(A-5)中的特征值分解只适用于方阵，而且只有当A的所有特征值不同时（无重复特征值），它才存在。相反，SVD对所有的矩阵都适用，不管是方阵或是矩形矩阵，且与矩阵的特征值的数量和类型（正数、实数、复数）无关。

特征值在分析涉及到重复乘A或求e^A的算法时很有用。奇异值主要用于矩阵特性的研究中。SVD在有些计算任务中很有用，如解秩亏的最小二乘问题。

A.5.1 svd函数

内置的svd函数实现了矩阵的SVD。它可以有以下4种调用形式：

```
S = svd(A)
S = svd(A,0)
[U,S,V] = svd(A)
[U,S,V] = svd(A,0)
```

这里，A是任意矩阵，S是方程(A.10)中的对角矩阵Σ，U和V是方程(A-10)中的酉矩阵（unitary matrix）。当第二个参数0作为输入传给svd时，结果就是SVD的简略形式。简略分解用于A是$m \times n$矩阵（$m>n$）的情况下，简略SVD给出了矩阵U的前n列和方阵Σ（$n \times n$）。注意当$m>n$时，A的完全SVD分解得出了Σ矩阵，该矩阵的最后$m-n$行全是0。

756

内置的svds函数用来计算一个稀疏矩阵的有限个奇异值。

附录B 稀疏矩阵

稀疏矩阵的零元比非零元个数要多很多。非零元占的比例非常小，这样使用更复杂的方法来存储和检索矩阵元素就会更有优势。

稀疏矩阵并没有引入线性代数的新数学特征，创建和使用这些稀疏矩阵是一个重要的实现问题，对此有一个专门的程序库。要从这些稀疏矩阵中获益，就需要使用特殊的数据结构来只存储非零元，这与传统的存储矩阵的方法不同，传统方法存储矩阵中的所有元素——不管其是否为零。因此，术语**稀疏矩阵**（*sparse matrix*）和**满矩阵**（*full matrix*）只是在存储方法意义上的划分。稀疏矩阵在数学意义上等价于满矩阵。

MATLAB为稀疏矩阵提供了内部的支持。一旦创建好稀疏矩阵，它就可以像满矩阵那样在MATLAB表达式中使用。但是，并非所有的内置函数对稀疏矩阵和满矩阵都适用。这种情况下，MATLAB提供了专门的稀疏矩阵的版本。这样，常常不用多少额外的工作就可以获得使用稀疏矩阵所带来的益处。本附录的目的是介绍稀疏矩阵的存储思想和MATLAB内置的稀疏矩阵格式。要想了解额外信息，包括MATLAB中稀疏矩阵的内部表示细节，读者可以参考*Using* MATLAB[73]和Gilbert et al.[27]的文章。

B.1 对存储空间和浮点操作次数的节省

考虑使用稀疏矩阵技术所带来的节省存储空间的好处。令δ为一个稀疏矩阵的存储密度：

$$\delta=\frac{\text{非零元素数}}{\text{总元素数}} \tag{B-1}$$

例如，$n\times n$三对角矩阵的存储密度为

$$\delta=\frac{3n-2}{n^2}\approx\frac{3}{n}$$

当n增加时，存储密度会减小，或**稀疏度**（*sparsity*）会增加。只存储矩阵的非零元，就可确保计算机内存的使用量最少，这可以使一个很大的矩阵能够存储在给定的有限内存中。

除了减小内存需求，使用稀疏存储还可获得很显著的计算优势。当矩阵作为满矩阵存储时，进行一次矩阵-向量乘积需要$O(n^2)$量级的浮点操作次数；而当以稀疏格式存储时，一次矩阵-向量乘积只需要$O(n_{nz})$量级的浮点操作次数，其中n_{nz}是非零元的个数。对三对角矩阵来说，$n_{nz}\approx 3n$，故矩阵-向量乘积的计算量就可以从$O(n^2)$减小到$O(n)$。特别是当n很大时，操作上的节约是很显著的。

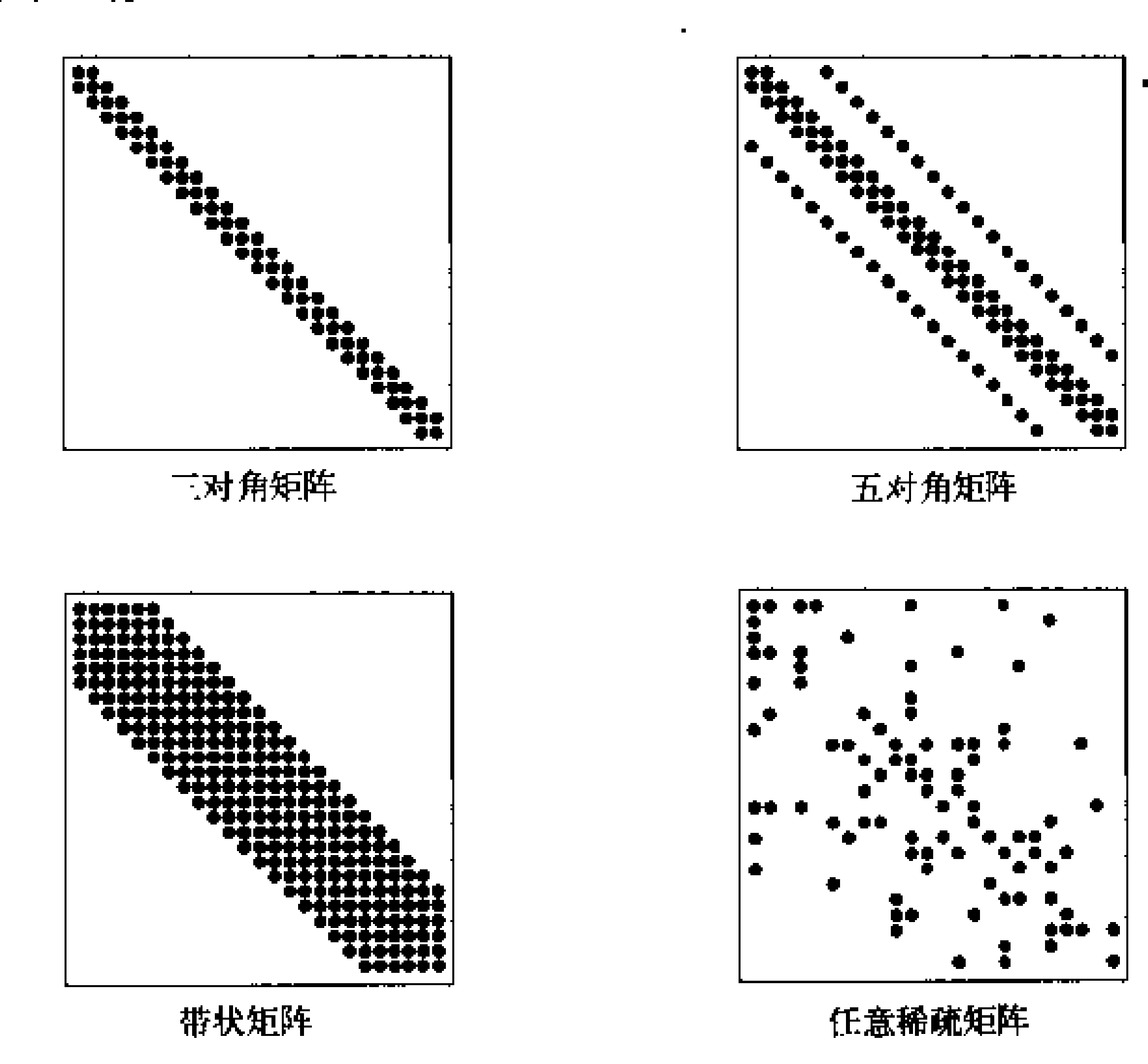
B.2 MATLAB中的稀疏矩阵格式

三对角矩阵是对角结构稀疏矩阵的一个特殊类型。稀疏矩阵其他的例子如图B-1所示。这包括五对角（pentadiagonal）及五对角以上的对角矩阵、带状矩阵（band matrix）和非零元任意分布的稀疏矩阵。

使用稀疏矩阵格式时，认识到有一些元素存储值可能为零这一点特别重要。例如，在特

殊情况下，需要对算法输入数据时，有可能要用到零元。其实，不必保证存储在稀疏矩阵数据结构中的所有元素值都为非零值。主要区别在于问题的构成，例如求偏微分方程数值解这样的问题，矩阵中会有一大部分元素总是为零。

758



图B-1 稀疏矩阵的四个例子：三对角矩阵、五对角矩阵、带状矩阵和非零元素任意分布矩阵。图中的非零元由实点表示

B.2.1 创建稀疏矩阵

MATLAB不会自动创建稀疏矩阵。但是，一旦创建稀疏矩阵，它就可像满矩阵一样使用。表B-1列出了一些内置的稀疏矩阵函数。在MATLAB命令提示符状态下键入`help sparfun`会得到其他函数的列表。

759

表B-1 基本稀疏矩阵函数

用于创建稀疏矩阵的函数:	
`spalloc`	为一个稀疏矩阵分配内存，然后通过调用其他稀疏矩阵函数来定义
`sparse`	将一个满矩阵转变成稀疏存储格式，或用由下标和元素表示的向量来组成稀疏矩阵
`spdiags`	由对角线元素来组成稀疏矩阵，或从一个稀疏矩阵中析取出对角线元素
`speye`	创建稀疏单位矩阵
`sprand`	由随机元素创建一个稀疏矩阵
用于稀疏矩阵操作的函数	
`full`	将一个稀疏矩阵转换成满矩阵
`nnz`	返回稀疏矩阵或满矩阵中非零元的个数
`nonzeros`	析取一个稀疏矩阵中的非零元。其结果为一个向量
`nzmax`	返回分配给一个稀疏矩阵的最大非零元个数。一般情况下，`nzmax(A)` ≥ `nnz(A)`

（续）

spparms	设定稀疏矩阵的参数，这些参数用于控制其他稀疏矩阵程序对这个矩阵如何操作
spones	用1替换稀疏矩阵中所有非零元
spy	将稀疏矩阵中的非零元表示成二维数组中对应的点来画出矩阵的形象化图示

MATLAB中任意稀疏矩阵可通过指定非零元的行下标和列下标（i和j）及元素值列表来创建。因此，稀疏矩阵可由一个表来描述，其中表的每一行包括下标i和j以及元素值$a_{i,j}$。例如，矩阵

$$A=\begin{bmatrix}1 & 0 & 1\\ 0 & 2 & 0\\ 3 & 3 & 0\end{bmatrix} \tag{B-2}$$

就可以用下表来表示：

行下标	列下标	元素值
1	1	1
3	1	3
2	2	2
3	2	3
1	3	1

未在上表中列出的所有元素就被认为是零，MATLAB不存储这些零值。内置的sparse函数可由这样一个描述矩阵元素的表来建立一个稀疏矩阵。

函数sparse的调用方法如下：

```
S = sparse(A)
S = sparse(is,js,as)
S = sparse(is,js,as,nrows,ncols)
S = sparse(is,js,as,nrows,ncols,nonzeros)
```

760

这里，S是要创建的稀疏矩阵。在sparse的第一种调用形式中，输入A是一个满矩阵，因为需要首先创建满矩阵A，所以sparse的这种用法在节约空间方面并不很有效。但是，此法可用于调试稀疏矩阵的计算中，当计算要使用稀疏矩阵的满矩阵存储的代码存在时，从一个满矩阵直接创建一个稀疏矩阵是很有用的。

在sparse函数的第二种调用形式中，*is*、*js*和*as*三个向量包含了稀疏矩阵表格表示中的三列数据。可选参数*nrows*和*ncols*分别明确地定义了S中的行数和列数。如果没有指定*nrows*和*ncols*，就使用m = max(*is*)和n = max(*js*)来创建$m \times n$稀疏矩阵。如果可选的*nonzeros*参数未指定，那么就通过用足够的存储空间存储length(*as*)个非零元来创建S。这很有用，例如，在sparse调用之前，所有创建S需要的元素都已知。在一些应用中，例如，在组合一个有穷元素模型的刚度矩阵（stiffness matrix，或称劲度矩阵）时，是一次添加一小组元素来构造矩阵S的。这种情况下可选参数*nonzeros*就可用来告诉sparse函数分配足够的空间来存储最终的S。

有时，首先通过创建由下标和元素表示的向量来创建稀疏矩阵并不方便。这种情况下，就可以用spalloc函数来创建一个空的稀疏矩阵，矩阵的元素可以通过每次添加入一个元素

或一次添加一组元素的赋值操作来得到。函数spalloc的句法为:

```
S = spalloc(m,n,nonz)
```

其中m和n是行数和列数，$nonz$是S中非零元的总个数。$nonz$的值不需要精确地知道。如果$nonz$小于存储矩阵需要的非零元的最终数目，MATLAB就会根据需要为S分配更多的存储空间。如果$nonz$比所需的大，为非零元预留的多余存储空间就不会被使用。为了取得最大的计算效率，在首次为矩阵分配内存时宁可过大、也不要过小。

例B.1　创建和显示一个简单的稀疏矩阵

方程(B-2)定义的矩阵**A**的一个稀疏形式可由以下语句创建:

```
>> is = [ 1 3 2 3 1];
>> js = [ 1 1 2 2 3];
>> as = [ 1 3 2 3 1];
>> S = sparse(is,js,as)
S =
   (1,1)        1
   (3,1)        3
   (2,2)        2
   (3,2)        3
   (1,3)        1
```

761

一个稀疏矩阵的缺省显示形式是：一个由非零元的位置和元素值所组成的表格。每一行的输出对应于稀疏矩阵中的一个元素。圆括号中的数字是元素的行下标和列下标，最后的数字是元素的值。

和满矩阵一样，S中的单个元素可用下标记号来引用和赋值。

```
>> S(1,3)
ans =
     1
```

虽然S(1,2)的值并未存储——它是在稀疏矩阵存储策略中跳过的那些零中的一个——但是它的值也可以用下标记号返回。

```
>> S(1,2)
ans =
     0
```

函数full将一个矩阵从稀疏矩阵格式转变成满矩阵格式。它提供了一种将（小）稀疏矩阵显示为数组的便捷方法:

```
>> full(S)
ans =
     1  0  1
     0  2  0
     3  3  0
```

例B.2　使用spalloc函数创建一个稀疏矩阵

下面的语句使用spalloc函数来为稀疏的5×5三对角矩阵分配存储空间，然后一次加入一个元素。

```
S = spalloc(5,5,13);        % 5 x 5 matrix with 13 nonzero elements
S(1,1) = 2; S(1,2) = -1;    % first row
for i=2:4
```

```
  S(i,i-1) = -1; S(i,i) = 2; S(i,i+1) = -1;
end
S(5,4) = -1; S(5,5) = 2;   % last row
```

内置的nnz和nzmax函数分别返回稀疏矩阵中非零元的实际个数和最大个数。例如，最大个数可能比前面例子中创建的三对角矩阵非零元的实际个数大。

```
>> S = spalloc(5,5,18);   % 5 x 5 sparse matrix with room for 18 nonzeros
>> S = ...                % assign tridiagonal entries as above
>> nnz(S)
ans =
    13
>> nzmax(S)
ans =
    18
```

762

在用稀疏格式创建S后，可以向其中添加更多的元素。这样并不会改变S的类型——它依然是稀疏的

```
>> S(1,5) = -1;    S(5,1) = -1;
>> nnz(S)
ans =
    15

>> nzmax(S)
ans =
    18
```

为得到最大的代码效率（最少的执行时间），应该避免采用一次添加一个元素的办法来建立稀疏矩阵，除非在元素添加前已经为整个稀疏矩阵分配了足够的存储空间。在稀疏矩阵中，当需要分配额外空间来插入元素时，MATLAB需要重新组织存储矩阵的内部数据结构。这样对较大规模的稀疏矩阵来说，重新组织数据结构所消耗的CPU时间就变得很大。建立稀疏矩阵最有效的过程就是首先创建行下标向量、列下标向量和元素值，然后如前面示范的那样，调用S = sparse(is,js,as)来组合稀疏矩阵。

对角结构的稀疏矩阵 函数spdiags用来创建和析取稀疏矩阵的对角线元素。它对应于满矩阵的diag函数，但这两个函数的输入参数明显不同。使用

```
sparseMatrix = spdiags(diagElements, diagIndices, nrows, ncols)
```

来创建稀疏矩阵。其中*diagElements*是一个*nrows* × *p*矩阵，该矩阵的元素要插入到*sparseMatrix*中，*p*是有非零元的对角线的总数。参数*diagIndices*是长度为*p*的行向量，用来指示*diagElements*的每列分布在*sparseMatrix*对角线上的位置。*sparseMatrix*由*nrows*行和*ncols*列创建，对角线数*p*不用明确地指定，它由*diagElements*和*diagIndices*中的列数定义。

763

使用spdiags函数，三对角矩阵（-1，2，-1)的稀疏矩阵表示可由下列语句创建。

```
>> n = 5;                  % Size of the n-by-n matrix to create
>> a = 2*ones(n,1);        % Main diagonal entries
>> b = -ones(n,1);         % Sub- and super-diagonal entries
>> S = spdiags([b a b],[-1 0 1],n,n);
```

对角元素由表达式[b a b]组合成$n \times 3$矩阵，表达式[-1 0 1]中的-1表示把b存储在主对角线下方并与主对角线平行的位置上，+1表示把b存储在主对角线上方并与主对角线平行

的位置上，a将存储在主对角线上。

例B.3 MATLAB任意三对角矩阵的稀疏格式

假设a、b、c是MATLAB列向量，这些列向量有n个元素。下面的语句使用方程(7-31)中的记号建立一个三对角矩阵：

```
a = ..., b = ...; c = ...;        % define a, b, and c
n = length(a);                    % must equal length(b) and length(c)
A = spdiags([ [c(2:n); 0] a [0; b(1:n-1)] ], [-1 0 1], n,n);
```

函数spdiags的第一个参数是有三列的矩阵：

$$\begin{bmatrix} c_2 & a_1 & 0 \\ c_3 & a_2 & b_1 \\ \vdots & \vdots & \vdots \\ c_n & a_{n-1} & b_{n-2} \\ 0 & a_n & b_{n-1} \end{bmatrix}$$

它是通过变换向量b和c中的元素位置得到的。函数spdiags将输入矩阵的元素按列插入到稀疏矩阵中，因此需要通过变换来创建方程(7-31)中的矩阵。

程序清单B-1中的tridiags函数提供了创建稀疏三对角矩阵的额外自动操作。此函数在NMM工具箱的linalg目录下。使用此函数，矩阵的对角线可以指定为向量或标量。如果输入是标量，标量值就沿着稀疏矩阵的对角线排列。

下面的语句比较了创建稀疏三对角矩阵时两种解释输入向量的方法：

```
>> a = 1:5; b = a; c = a;
>> A = tridiags(5,a,b,c);
>> full(A)
ans =
     1     1     0     0     0
     2     2     2     0     0
     0     3     3     3     0
     0     0     4     4     4
     0     0     0     5     5
```



```
>> a =(1:5)' ;b = a; c = a;
>> B = spdiags([c a b],[-1 0 1],5,5);
>> full(B)
ans =
     1     2     0     0     0
     1     2     3     0     0
     0     2     3     4     0
     0     0     3     4     5
     0     0     0     4     5
```

程序清单B-1 函数tridiags使用MATLAB内置的稀疏格式来创建稀疏三对角矩阵

```
function A = tridiags(n,a,b,c)
% tridiags  Create sparse tridiagonal matrix from two or three scalars or
%           vectors
%
% Synopsis:  A = tridiags(n,a,b)
%            A = tridiags(n,a,b,c)
%
% Input:     n     = size of desired (n by n) matrix
%            a,b,c = scalar or vector values for the main, super and sub
%                    diagonals.  If a, b, or c are scalars, they are
%                    replicated to fill the diagonals.  If c is not supplied,
%                    c = b is assumed
%
% Output:    A = sparse, tridiagonal matrix of the form
%
```

```
%               a  b  0  0             a(1)  b(1)   0     0
%               c  a  b  0     or      c(2)  a(2)  b(2)   0
%               0  c  a  b     ,        0    c(3)  a(3)  b(3)
%               0  0  c  a              0     0    c(4)  a(4)

if nargin<4,  c = b;  end

if length(a)==1,  a = a*ones(n,1);  end    %  replicate scalars
if length(b)==1,  b = b*ones(n,1);  end
if length(c)==1,  c = c*ones(n,1);  end
if length(a)<n | length(b)<n | length(c)<n
  error(sprintf('a,b,c must be scalars or have length >= %d',n));
end

a = a(:);  b = b(:);  c = c(:);   %  guarantee column vectors
A = spdiags([ [c(2:n); 0] a [0; b(1:n-1)] ],[-1 0 1], n,n);
```

既然$A = B^T$，就可以使用下列语句避免tridiags函数中所使用的变换列的方法：

```
A = spdiags( [c a b], [-1 0 1], 5, 5)';
```

将spdiags的输出进行转置，需要更详细地了解函数的工作细节。这不如使用低级一些的tridiags函数来得明显。

765

B.2.2 稀疏矩阵的运算

一旦稀疏矩阵定义好，就能像其他矩阵一样地使用它。MATLAB知道如何处理包含满矩阵和稀疏矩阵的混合代数问题。例如，考虑如下关于例B.1中所创建的稀疏矩阵的MATLAB程序段。

```
>> S = sparse(...);   % see Example B.1
>> A = full(S);
>> A-S
ans =
     0     0     0
     0     0     0
     0     0     0
```

混合矩阵加减的结果总是一个满矩阵。涉及两个稀疏矩阵的操作的结果一般仍是稀疏矩阵，MATLAB会尽量保持其稀疏性。

```
>> B = S/2;
>> C = S-B
C =
   (1,1)       0.5000
   (3,1)       1.5000
   (2,2)       1.0000
   (3,2)       1.5000
   (1,3)       0.5000
```

例B.4 稀疏存储对计算量的节省

函数splint就使用了稀疏存储，它创建数据对集(x,y)的一个样条插值（参见10.3.5节）。要得到样条的系数，需要解一个三对角方程组。在本例中我们示范使用稀疏格式来存储三对

角矩阵所带来的计算量的节省。

作为比较的基础，函数splintFull实现了与splint相同的数值任务，但是它使用的是系数矩阵的满矩阵存储。函数splintFull在此处未列出，但包含在NMM工具箱的interpolate目录下。下面的语句描述了splint函数（左边）和splintFull函数（右边）不同的代码行。

766

稀疏存储

```
% --- Set up system of equations
...
A = tridiags(n,beta,alpha,gamma);
...
% --- Solve the system for b
mmdflag = spparms('autommd');
spparms('autommd',0);
b = A\delta;
spparms('autommd',mmdflag);
```

满存储

```
% --- Set up system of equations
...
A = tridiag(n,beta,alpha,gamma);
...
% --- Solve the system for b
b = A\delta;
```

NMM工具箱中的tridiags函数和tridiag函数分别创建了稀疏和满三对角矩阵。对spparms函数的调用取消了在解稀疏方程组之前一般要进行的行重新排序。行重新排序的目的是试图减少LU分解中的填充。既然此问题的系数矩阵是三对角矩阵，L和U因子就不会由于填充而增加额外的存储，所以取消行重新排序就在不影响浮点操作次数的前提下节省了工作量。

函数compSplintFlops（包含在NMM工具箱的interpolate目录下，此处未列出）比较了使用splint和splintFull函数对数据对集(x,y)进行插值的浮点操作次数。计算所需的工作量作为定义样条的结点数的函数来测量。运行compSplintFlops得到如下结果：

```
>> compSplintFlops
```

knots	sparse flops	full flops	relative effort
4	214	292	0.7329
8	401	859	0.4668
16	772	2838	0.2720
32	1511	10281	0.1470
64	2986	39020	0.0765
128	5933	151599	0.0391
256	11824	598066	0.0198
512	23603	2376245	0.0099

第一列是建立样条的结点数，第二和第三列是建立和计算样条插值的浮点操作次数，第四列是用稀疏存储方法的浮点操作次数和用满存储方法的浮点操作次数的比率。虽然splint和splintFull函数作了其他的工作，但是在求样条插值代码的浮点操作次数时，求解系数矩阵还是主要的工作。很明显，随着问题大小的增加，用稀疏矩阵格式显著地节省了计算量。

767

参考文献

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1965.

[2] F. S. Acton. *Numerical Methods That Work*. The Mathematical Association of America, Washington, D.C., 1990.

[3] T. J. Akai. *Applied Numerical Methods for Engineers*. Wiley, New York, 1994.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorenson. *LAPACK User's Guide*. SIAM, Philadelphia, 1992.

[5] B. M. Ayyub and R. H. McCuen. *Probability, Statistics, and Reliability for Engineers*. CRC Press, Boca Raton, FL, 1997.

[6] P. R. Bevington and D. K. Robinson. *Data Reduction and Error Analysis for the Physical Sciences*. McGraw-Hill, New York, 1992.

[7] A. Björk. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[8] R. P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[9] R. L. Burden and J. D. Faires. *Numerical Analysis*. PWS Publishers, New York, 3d edition, 1985.

[10] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing*. Brooks/Cole Publishing Co., Pacific Grove, CA, 3d edition, 1994.

[11] S. D. Conte and C. de Boor. *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill, New York, 3d edition, 1980.

[12] B. N. Datta. *Numerical Linear Algebra and Applications*. Brooks/Cole, Pacific Grove, CA, 1995.

[13] P. J. Davis and P. Rabinowitz. *Methods of Numerical Integration*. Academic Press, New York, 2d edition, 1984.

[14] C. de Boor. *A Practical Guide to Splines*. Springer-Verlag, New York, 1978.

[15] J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[16] J. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. SIAM, Philadelphia, 1996.

[17] P. Dierckx. *Curve and Surface Fitting with Splines*. Clarenden Press, Oxford, 1993.

[18] J. Dormand and P. Prince. A family of embedded Runge–Kutta formulae. *Journal of Computational and Applied Mathematics* **6**:19–26, 1980.

[19] N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley, New York, 3d edition, 1998.

[20] A. Edelman. The mathematics of the Pentium division bug. *SIAM Review* **39(1)**:54–67, Mar 1997.

[21] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design: A Practical Guide*. Academic Press, Inc., Boston, 2d edition, 1990.

[22] L. V. Fausett. *Applied Numerical Analysis Using* MATLAB. Prentice Hall, Upper Saddle River, NJ, 1999.

[23] B. A. Finlayson. *Nonlinear Analysis in Chemical Engineering*. McGraw-Hill, New York, 1980.

[24] G. Forsythe, M. Malcolm, and C. Moler. *Computer Methods for Mathematical Computations*. Prentice-Hall, Englewood Cliffs, NJ, 1977.

[25] W. Gander and W. Gautschi. Adaptive Quadrature—Revisited. *BIT*, **40(1)**:84–101, 2000.

[26] W. Gautschi. On the construction of Gaussian quadrature rules from modified moments. *Mathematics of Computation*, **24(110)**:245–260, Apr. 1970.

[27] J. R. Gilbert, C. Moler, and R. Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications* **13(1)**:333–356, Mar 1992.

[28] P. E. Gill, W. Murray, and M. H. Wright. *Numerical Linear Algebra and Optimization*, volume 1. Addison-Wesley, Redwood City, CA, 1991.

[29] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys* **23(1)**:5–48, March 1991.

[30] H. H. Goldstine. *The Computer: from Pascal to von Neumann*. Princeton University Press, Princeton, NJ, 1972.

[31] G. Golub and J. M. Ortega. *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. Academic Press, Inc., Boston, 1992.

[32] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, MD, 3d edition, 1996.

[33] G. H. Golub and J. H. Welsch. Calculation of Gauss quadrature rules. *Mathematics of Computation* **23**:221–230, 1969.

[34] D. Hanselman and B. Littlefield. *Mastering* MATLAB *5: A Comprehensive Tutorial and Reference*. Prentice Hall, Upper Saddle River, NJ, 1998.

[35] M. T. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, New York, 1997.

[36] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, 1996.

[37] R. R. Hocking. *Methods and Applications of Linear Models: Regression and the Analysis of Variance*. Wiley, New York, 1996.

[38] R. A. Horn and C. R. Johnson. *Matrix Analysis*. Cambridge University Press, Cambridge, 1985.

[39] F. P. Incropera and D. P. DeWitt. *Introduction to Heat Transfer*. Wiley, New York, 2d edition, 1990.

[40] E. Isaacson and H. B. Keller. *Analysis of Numerical Methods*. Dover, New York, 1994.

[41] M. Jenkins and J. Traub. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration. *Numerical Mathematics* **14**:252–263, 1970.

[42] M. Jenkins and J. Traub. Zeros of a complex polynomial. *Communications of the ACM* **15(2)**:97–99, Feb 1972.

[43] D. Kahaner, C. Moler, and S. Nash. *Numerical Methods and Software*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[44] Kelley. *Iterative Methods for Linear and Nonlinear Equations*. SIAM, Philadelphia, 1995.

[45] Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.

[46] B. W. Kernighan and R. Pike. *The Practice of Programming*. Addison-Wesley, Reading, MA, 1999.

[47] B. W. Kernighan and P. J. Plauger. *The Elements of Programming Style*. McGraw-Hill, New York, 2d edition, 1978.

[48] A. R. Krommer and C. W. Ueberhuber. *Computational Integration*. SIAM, Philadelphia, 1998.

[49] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, Philadelphia, 1995.

[50] G. Lindfield and J. Penny. *Numerical Methods Using* MATLAB. Ellis Horwood, New York, 1995.

[51] P. Marchand. *Graphics and GUIs with* MATLAB. CRC Press, Boca Raton, FL, 2d edition, 1999.

[52] R. L. Mason, R. F. Gunst, and J. L. Hess. *Statistical Design and Analysis of Experiments: with Applications to Engineering and Science*. Wiley, New York, 1989

[53] J. H. Mathews and K. D. Fink. *Numerical Methods Using* MATLAB. Prentice Hall, Upper Saddle River, NJ, 3d edition, 1999.

[54] C. Moler. A tale of two numbers. *SIAM News* **28(1)**: 1,16. Jan 1995.

[55] C. Moler. Floating points: IEEE standard unifies arithmetic model. MATLAB *News and Notes*, Fall 1996:11–13, 1996.

[56] C. Moler. Golden ODEs: New ordinary differential equation solvers for MATLAB and SIMULINK. MATLAB *News and Notes*, Summer 1996:11–13, 1996.

[57] F. C. Moon. *Chaotic and Fractal Dynamics: An Introduction for Applied Scientists and Engineers*. Wiley, New York, 1992.

[58] B. N. Parlett. *The Symmetric Eigenvalue Problem*. SIAM, Philadelphia, 1998.

[59] R. Piessens, E. de Doncker-Kapenga, C. Überhuber, and D. Kahaner. *QUADPACK. A Subroutine Package for Automatic Integration*. Springer-Verlag, Berlin, 1983.

[60] J. C. Polking. *Ordinary Differential Equations Using* MATLAB. Prentice Hall, Englewood Cliffs, NJ, 1995.

[61] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, New York, 2d edition, 1992.

[62] W. C. Reynolds. *Thermodynamic Properties in SI*. Stanford University, Department of Mechanical Engineering, Stanford, CA, 1979.

[63] T. P. Ryan. *Modern Regression Methods*. Wiley, New York, 1997.

[64] H. R. Schwarz. *Numerical Analysis: A Comprehensive Introduction*. Wiley, New York, 1989

[65] L. Shampine, R. Allen, Jr., and S. Preuss. *Fundamentals of Numerical Computing*. Wiley, New York, 1997.

[66] L. Shampine and M. W. Reichelt. The MATLAB ODE suite. *SIAM Journal on Scientific Computing* **18(1)**:1–22, Jan 1997.

[67] G. Stewart. Gauss, statistics, and Gaussian elimination. Technical Report TR-3307, University of Maryland, Department of Computer Science, College Park, Maryland, 1994.

[68] G. Stewart. *Afternotes on Numerical Analysis*. SIAM, Philadelphia, 1996.

[69] G. Stewart. *Matrix Algorithms, Volume I: Basic Decompositions*. SIAM, Philadel-

phia, 1996.

[70] J. Stoer and R. Bulirsh. *Introduction to Numerical Analysis*. Springer-Verlag, New York, 2d edition, 1993.

[71] G. Strang. *Linear Algebra and Its Applications*. Harcourt Brace Jovanovich, Fort Worth, TX, 3d edition, 1988.

[72] G. Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, Wellesley, MA, 1993.

[73] The Mathworks, Inc. Using MATLAB. The Mathworks, Inc., Natick, MA, 1996.

[74] The Mathworks, Inc. Using MATLAB Graphics. The Mathworks, Inc., Natick, MA, 1996.

[75] The Mathworks, Inc. MATLAB Language Reference Manual. The Mathworks, Inc., Natick, MA, 1996

[76] L. N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. SIAM, Philadelphia, 1997.

[77] C. F. Van Loan. *Introduction to Scientific Computing: A Matrix Vector Approach Using* MATLAB. Prentice Hall, Upper Saddle River, NJ, 1997.

[78] D. S. Watkins. *Fundamentals of Matrix Computations*. Wiley, New York, 1991.

[79] H. B. Wilson and L. H. Turcotte. *Advanced Mathematics and Mechanics Applications Using* MATLAB. CRC Press, Boca Raton, FL, 2d edition, 1997.

MATLAB工具箱函数

NMM工具箱m文件函数列表

索　引

索引中的页码为英文原书页码，与书中边栏的页码一致。

［Ｇｅｎｅｒａｌ Ｉｎｆｏｒｍａｔｉｏｎ］
书名＝数值方法和ＭＡＴＬＡＢ实现与应用
作者＝
页数＝５５４
ＳＳ号＝１１３１９３１８
出版日期＝